International Journal of Foundations of Computer Science
Vol. 33, No. 2 (2022) 91–118
© World Scientific Publishing Company
DOI: 10.1142/S0129054122500010

The Power of Machines That Control Experiments

Vasco Boavida De Brito
Department of Mathematics, Instituto Superior Técnico
Universidade de Lisboa, Portugal
vascobbrito@gmail.com

José Félix Costa*
Department of Mathematics, Instituto Superior Técnico
CFCUL – Centro de Filosofia das Ciências da Universidade de Lisboa
Universidade de Lisboa, Portugal
jose.felix.costa@tecnico.ulisboa.pt

Diogo Poças
LASIGE, Departamento de Informática, Faculdade de Ciências
Universidade de Lisboa, Portugal
dmpocas@fc.ul.pt

Received 22 February 2021
Accepted 12 January 2022
Published 16 February 2022
Communicated by Martin Kutrib

We consider the experimenter (e.g. the experimental physicist) as a Turing machine — the digital component — and the experiment of measurement — the analog component — as an oracle to the Turing machine. The algorithm running in the machine abstracts the experimental method of measurement (encoding the recursive structure of experimental actions) chosen by the experimenter. In this paper we prove that the central analogue-digital complexity classes $\mathtt{P}$, $\mathtt{P/poly}$ and $\mathtt{P/poly} \cap \mathtt{REC}$ can be characterized in terms of protocols to perform measurements controlled by standard Turing machines.

Keywords: Analogue computation; analogue-digital systems; hybrid systems; non-uniform complexity; Church-Turing thesis.

1. Introduction

An oracle Turing machine consists in a standard Turing machine coupled with a set — the oracle. Oracle Turing machines have an additional tape, the query tape, and additional distinguished states, such as the so called query state QUERY. Upon entering the query state, the machine makes a transition either to the distinguished state YES or to the distinguished state NO, depending if the content of the query

*Corresponding author.

tape is an element in the oracle set or not. This is done in a single time step. Oracle Turing machines decide non-decidable sets relative to adequate non-decidable oracle sets. Oracles however are external devices to the Turing machine that work as black boxes, and in the years since their first idealization by Alan Turing the question if there could be some computing model based in current physical theories that could break the Turing barrier emerged, even if only abstractly, or in ideal conditions.

A notable contribution to the subject is the Analogue Recurrent Neural Network (`ARNN`) introduced by Hava Siegelmann (see [25–28]), which consists in a neural network with real valued synaptic weights. In [26, 27], it is shown that the `ARNN` model decides sets that are not decidable by standard Turing machines. Questions regarding the feasibility of these machines arose, even though Younger et al. have claimed to have engineered an implementation of this model, the Optical `ARNN` (see [25, 31]), allegedly capable of producing outputs which display behavior compatible with that of chaotic systems.

A particular critic on any physical implementation of the `ARNN` model was Martin Davis (see [20, 21]). He argued that the reason why such a model is able to decide super-Turing sets is because the same sets are provided with the non-computable real weights in the first place. Furthermore, Davis reasonably claims that even if a machine could output a non-computable sequence of natural numbers, its non-computability could never be verified in finite time. One way of formalising Davis' argument is to interpret the computation of an `ARNN` as the process engaged by a computational system that, given some input w, performs a measurement $\mu \in \mathbb{R}$ of some concept (e.g. the strength of a synapse in a neural network) up to an accuracy depending on the size of w, and then decide if w is to be accepted by means of μ seen as advice. The number k of bits of precision required can be obtained in linear time in k. In this sense, the `ARNN` departs from being a realistic physical model in that the activation function of the neurons is stepwise linear with discontinuous derivatives. With a more realistic (analytic) activation function of the neurons, the time to read the next bit of a real weight is exponential in the number of bits already extracted. Although the Theory of Measurement (see [22, 24, 29]) does not take into account the physical time needed for a measurement of increasing precision (as a function of precision), the time complexity of a measurement is related with the computational extra power added by the real numbers that come into the computations. Measurements should be regarded as information with possible error that take time to accomplish. According to [10], this reduction of super-Turing capabilities can be so great that the real numbers add no more power than the rational numbers.

A different computational paradigm to that of Siegelmann and Sontag was introduced by Beggs, Tucker and Costa (starting with papers [4, 5]) in which Turing machines may communicate with some physical apparatus, and thus measure some unknown quantity, behaving as empirical scientists in a lab. The Turing machine therefore uses a physical experiment, based in some theory of classical physics, as an oracle, similarly to oracle Turing machines, and can execute queries over it. However, the execution of physical experiments takes more than some fixed time, thus restrictions have to be imposed over how much time the Turing machine waits for the physical experiment before cutting it off. The amount of time the machine waits has been given by a time constructible function — the time schedule —, which increases strictly with the size of the query. In most literature, time schedules with exponential growth had to be chosen (see [2, 14]). This computational model has two components at its disposal: the Turing machine, which corresponds to the digital part; and the physical experiment, which is the analogue part. This is why machines of this kind are known as analogue-digital Turing machines, or ADTM's. Multiple physical experiments have been considered for the physical oracle of ADTM's; the balance in [7], the collider in [12, 14], the Wheatstone bridge in [13], among others. The complexity classes involved in such computations bounded to a polynomial number of steps were fully characterised in [1, 2, 4–14, 16–18]. In here, we consider the simplest experiment, the smooth scatter experiment (SmSE), which is based on a sub-theory of Newtonian mechanics, although any other physical experiment could have been used to obtain the same results. We call an ADTM with this physical oracle a smooth scatter machine (SmSM).

The SmSE is described in two dimensional space (see Fig. 1). It consists of two main components: a smooth symmetrical wedge with vertex in some fixed position between 0 and 1; and a cannon pointed towards the wedge, which can shoot particles upon request of the SmSM, forcing these particle to be bounced off this barrier. The experiment also has sensors on either side of the wedge which are activated if a shot particle crosses them, effectively allowing us to determine on which side of the vertex the particle hit the wedge. With information on the outcomes of successive shots we can therefore carry out measurements over the position of the wedge, and obtain approximations of this value. The position where the cannon fires the particle is determined by the digital part of the SmSM, the Turing machine. The machine writes the desired position in the query tape and makes a transition to the query state, just like oracle Turing machines. The exact position where the cannon fires is determined by the communication protocol between the analogue and digital parts of the SmSM. We have been considering two such protocols: the infinite precision protocol, where the cannon fires the particle exactly in the position written in the query tape; and the fixed precision protocol where upon entering the query state with the number z written in the query tape, the cannon fires somewhere in the interval $[z - \varepsilon, z + \varepsilon]$, for a fixed ε, which confers a degree of uncertainty to computations over the SmSM.[a]

In this paper, we develop further results regarding the the power of analogue-digital computation. The paper is structured in three main sections. In Sec. 2, we formally introduce SmSM's in detail. Note that the SmSM is a very simple exercise

[a]It is interesting to note that in the 50's von Neumann was already studying the possibility of harnessing continuous values present in Nature, and of carrying out reliable computation using unreliable components, either from artificial or naturally occurring systems ([30]). Likewise, although the systems are very different, in the work that follows it is required to carry out computations using unreliable procedures also.

in measurement and that other gedankenexperimente involving measurement, as far as we investigated in the last decade, turn out to be equivalent to the SmSM in what concerns the complexity of the measurement map. Any attempt to consider more sophisticated measurements only contributes to the illegibility of the paper. In Sec. 3, we discuss some previous results in order to contextualize the work ahead. Finally, in Sec. 4, we advance a new characterisation of relevant classes of sets for analogue-digital computation such as P, P/poly and P/poly $\cap$ REC based on non-computable time schedules.[b] The main statements of this paper can be synthesised as follows:

Theorem 1. *Let IN denote the class of increasing total functions, CI the class of computable increasing total functions and TC the class of all strictly increasing time constructible functions. Let AP$(\mathcal{F})$ be the class of sets decidable by some SmSM clocked in polynomial time, using a time schedule from $\mathcal{F}$. We have that:*

$$\begin{aligned} \mathit{AP}(\mathit{TC}) &= \mathit{P} \\ \mathit{AP}(\mathit{IN}) &= \mathit{P/poly} \\ \mathit{AP}(\mathit{CI}) &= \mathit{P/poly} \cap \mathit{REC}. \end{aligned}$$

Comparing with the ARNN model (as in [26]), we conclude that while in the ARNN model clocked in polynomial time by changing the type of the weights from rationals to reals and from reals to computable reals, we get the classes P, P/poly, and P/poly $\cap$ REC, in our new setting by changing the type of time schedules from TC to IN and from IN to CI, we get the same classes P, P/poly, and P/poly $\cap$ REC.

2. Physical Experiments as Oracles

In line with the work developed in [1, 2, 4–14, 16–18], inter alia, our main focus is the study of the *analogue-digital Turing machine*, that is, a *Turing machine* that besides being a computational device, also behaves as an algorithmic experimenter performing analogue physical measurements. As we shall see, the introduction of these experiments boosts the computational power of the machine to super-Turing levels. Although we can imagine these measurements as oracle consultations, as we shall see, these experiments work in a different manner, allowing us to include into the computations of the Turing machine successive timed approximations to infinite precision real numbers (denoted by $\mathbb{R}$ as usual).

2.1. *Analogue-digital Turing machines*

An analogue-digital Turing machine (ADTM), is a Turing machine with one input, one output and working tapes, as well as an additional query tape used to control and

[b] Let $\mathcal{C}$ be a class of sets and $\mathcal{F}$ a class of total functions of signature $\mathbb{N} \to \Sigma^\star$. The non-uniform class $\mathcal{C}/\mathcal{F}$ is the class of sets A for which some $C \in \mathcal{C}$ and some $f \in \mathcal{F}$ are such that, for every w, $w \in A$ if and only if $\langle w, f(|w|)\rangle \in C$. If we take $\mathcal{C}$ as P and $\mathcal{F}$ as poly, the set of polynomials over $\mathbb{N}$, then we get class P/poly. The class of recursive sets is herein denoted by REC.

communicate with a given physical experiment. The finite control of each machine include some distinguished states, an initial state, accepting and rejecting states, a query state and a finite number of possible outcome states. When entering the query state, the machine waits for a response from the physical experimental apparatus, and resumes the computation in one of the outcome states. An `ADTM` is thus nothing more than the combination of three components: the **Turing machine**, that carries out the digital part of the computation; the **physical experiment**, that receives the queries from the Turing machine and carries out the experiments; and the **interface of communication** between these digital and analogue components. In a sense, these machines resemble the hybrid machines of the 60's, where the digital component was performed by a digital computer, e.g. solving the temporal term of Poisson's equation, the analog component implemented by an special purpose analog device, e.g. a membrane solving the Laplacian term of the same equation, and the interface replaced by a transducer (see [19]).

2.1.1. *The physical experiment*

Physical experiments are what will serve as physical oracles to the `ADTM`, providing the analogue component of our computation. In [2, 4, 5, 7, 9, 11, 14, 16, 18] a variety of such experiments have been considered, all with some characteristics in particular: The experiment considered must have some initial condition that can be varied (in a set $\mathcal{I}$), producing possibly different outcomes, of which there are only finitely many (we shall denote their set by $\mathcal{R}$). Furthermore, there must be a physical process underlying the experiment, fully explained by a physical theory. And finally, there must exist a time function $t_{exp} : \mathcal{I} \to \mathbb{R}$, which is the experimental time, over which we make the following assumptions:

(1) $\mathcal{I} = [0, 1]$;
(2) $\forall x \in \mathcal{I}$, $t_{exp}(x) > 0$;
(3) t_{exp} is strictly increasing and unbounded in $[0, y[$;
(4) t_{exp} is strictly decreasing and unbounded in $]y, 1]$;
(5) t_{exp} is continuous in $[0, y[\cup]y, 1]$.

The experimental time does not include the time to set up the physical experiment, we consider that this is done instantaneously. The experimental time reflects instead the time intrinsic to the physical phenomenon under investigation or measurement.

2.1.2. *The communication interface*

The communication interface manages the communication between the Turing machine and the physical experiment. It is composed of two parts: one is the **protocol**, which specifies to the physical experiment which sequence of instructions to carry out. It begins by reading the query word from the query tape of the Turing machine, and its final step is to evaluate the result of the experiment and carry out the computation of the Turing machine from a particular outcome state; the other component of the communication interface is the **time schedule**, that has been considered a time-constructible increasing total function $T : \mathbb{N} \to \mathbb{N}$ that limits the time that the Turing machine waits for an outcome of the experiment. We consider that if an experiment that is being carried out with a query of size n produces no result in time $T(n)$, then the protocol terminates the experiment and carries on the computation in a particular outcome state (further on we will always refer to this state as q_t, or timeout state). The inclusion of the time schedule is necessary because, as explained in Sec. 2.1.1, queries to the physical experiment may take an arbitrarily large, possibly infinite, amount of time. Limiting the time we wait for the experiment thus prevents the machine from never halting, allowing a study of the complexity classes that are decided by `ADTM`'s when considering different time schedules, even in the case they are non-computable (possibly controlled by the environment), while also limiting the precision of the experiment, as we shall see.

It is important to explain exactly what we mean when we say that the Turing machine "waits", and when we say that the experiment takes "time" greater than $T(n)$. When the Turing machine enters the query state, with a query word of size n, it simulates a clock with $T(n)$ time steps, which is possible when T is time-constructible, and carries on the computation as stated above. Saying that the machine "waits" is oversimplifying this fact, since it does not literally halt when an experiment is carried out, waiting for some external signal to tell it enough "time" has passed. It clocks itself with discrete time steps, corresponding to the time taken to carry out steps of the Turing machine. Next, we relate the real physical time taken by the experiment, t_{exp}, with the time taken by the Turing machine to carry out $T(n)$ steps, which we can do by making the units of these times the same. If T is not time-constructible, then the Turing machine undergoes a process of busy waiting until an answer to the query is received.

2.2. *Smooth scatter experiment*

We introduce now an example of a physical experiment — the Smooth Scatter Experiment — that is considered for the `ADTM`'s in this paper. We define this experiment just as it was done in [2, 16]. First of all, we need to define the **physical theory** upon which our experiment is based. In this case this theory is a sub-theory of Newtonian mechanics, in particular the following laws and assumptions:

(1) Particles obey the Newtonian laws of motion in the two-dimensional place;
(2) Collisions are perfectly elastic, i.e. kinetic energy is conserved in collisions;
(3) Barriers are rigid, hence they don't deform as a result of collisions;
(4) A cannon can project a particle with a given velocity in a given direction;
(5) Detectors are capable of detecting if a particle has crossed a given region of the plane;

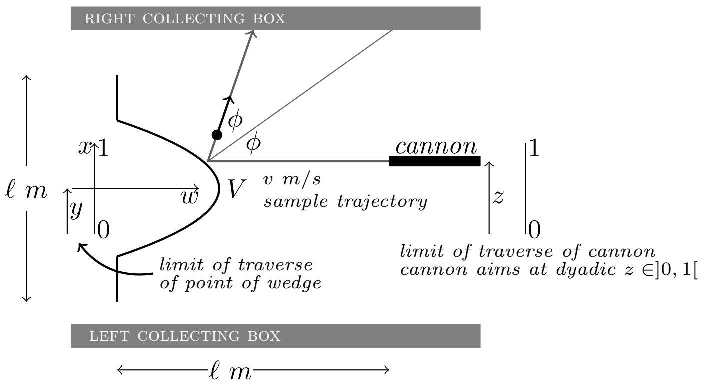

Fig. 1. The smooth scatter experiment.

With our physical laws and assumptions at hand, we can now describe the physical experiment that will be considered throughout this paper, the Smooth Scatter Experiment, or `SmSE`, as described in Fig. 1. Consider an horizontal and vertical axes, w and x respectively. We consider a smooth curve, symmetric relatively to some horizontal line with height $y \in [0, 1]$ which is unknown. Carrying out the experiment means firing a particle from a cannon at a fixed speed, perpendicularly to the x axis of the curve, at some position $z \in [0, 1]$. There are two sensors set in lines parallel to the w axis, one above, and another below the curve, and both at the same distance of the w axis, that detect if the fired particle has crossed them or not. If the fired particle crosses the upper sensor, then we say it enters the right collecting box, if it crosses the lower one, then we say in enters the left collecting box.

Thus, in whichever way we carry out the experiment, we only have three possible outcomes:

(1) $z > y$ and thus after some time, the fired particle is detected by the upper sensor;
(2) $z < y$ and thus after some time, the fired particle is detected by the lower sensor;
(3) $z = y$ and the fired particle is not detected by either sensor.

Thus if we know what the result of the experiment is, and carrying it out multiple times, we can effectively measure y (assuming of course that the experiment does not run forever, or that we can find a way to deal with that). The proof of the following proposition gives us the bounds for the experimental time.

Proposition 2 (Beggs, Costa and Tucker in [16]). *Let $g(x)$ be the function describing the shape of the wedge of a `SmSE`. Suppose that $g(x)$ is n times continuously differentiable near $x = 0$, all its derivatives up to the $(n-1)$th vanish at $x = 0$, and the nth derivative is nonzero. Then, when the `SmSE`, with vertex position* *at y, fires the cannon at position z, the time needed to detect the particle in one of the boxes is $t_{exp}(z)$, where:*

$$\frac{A}{|y-z|^{n-1}} \leq t_{exp}(z) \leq \frac{B}{|y-z|^{n-1}} \tag{1}$$

for constants $A, B > 0$ and for $|y-z|$ sufficiently small.

From now on, just as was done in [2], we will consider that the function $g(x)$ that describes the shape of the wedge is continuously differentiable twice, with $g'(y) = 0$ and $g''(y) \neq 0$, effectively having $n = 2$ in the equation above. If one wanted to measure y using the cannon position z with $|y-z| \leq 2^{-m}$, then from Eq. (1) we can conclude that $t_{exp}(z) \geq A.2^m$, thus the experimental time is at least exponential in the length / size of required precision over y, and infinite when $z = y$.

According to [8], any reasonable physical experiment could have been used:

> We are led to question and make a first conjecture that this is common to all physical experiments: For all reasonable physical theories $\mathcal{T}$, for all reasonable physical measurements based upon $\mathcal{T}$, the $\mathcal{T}$-time for the physical experiment is at least exponential in the size of the precision.

This conjecture comes from the fact that this exponential bound on experimental time has been observed in every reasonable physical experiment considered by Beggs, Costa and Tucker in, for instance, [2, 7, 9, 13, 14, 16].

2.3. *The smooth scatter machine*

A smooth scatter machine, or `SmSM`, is nothing more than an analogue-digital Turing machine which uses a smooth scatter experiment as a physical oracle. It communicates with the `SmSE` through the query tape. When the machine make a transition to the query state, the `SmSE` is carried out with the cannon in the position given by the dyadic rational in the tape. Since the `SmSE` can be carried out with cannon positions in the interval $[0, 1]$, and the `SmSM` can only print binary words in the query tape, we have that the `SmSM` can either write the word "1" or a binary word beginning with 0 in the query tape. Hence queries can be carried out with $z = z_0 z_1 \ldots z_n \in \{1\} \cup \{0s : s \in \{0,1\}^\star\}$, with the corresponding dyadic rational being:

$$\sum_{i=0}^{n} z_i 2^{-i}$$

This number represents the cannon position in $[0, 1]$ with which the experiment will take place. We will use z to denote both the query word used by the `SmSM` and the corresponding dyadic rational, for it is implied by context which form we are using.

Additionally, since a `SmSM` is an `ADTM`, to fully characterize it we need to define a time schedule T and a communication protocol, as described in Sec. 2.1.2. The three protocols that have been considered so far in the previous papers are:

Protocol 1. *We say that a* `SmSM` $\mathcal{M}$, *with access to a* `SmSE`, *has an* error-free protocol *or an* infinite precision protocol *if when* $\mathcal{M}$ *enters the query state with the word* z *in the query tape*, *the* `SmSE` *carries out the experiment with the cannon in the position* $z' = z$.

Protocol 2. *We say that a* `SmSM` $\mathcal{M}$, *with access to a* `SmSE`, *has an* unbounded precision protocol *if when* $\mathcal{M}$ *enters the query state with the word* z *in the query tape, the* `SmSE` *carries out the experiment with the cannon in the position* $z' \in [z - 2^{-|z|}, z + 2^{-|z|}]$, *where* $|z|$ *is the number of bits of* z. *The probability distribution of* z' *in this interval is considered to be uniform and independent of previous experiments.*

Protocol 3. *We say that a* `SmSM` $\mathcal{M}$, *with access to a* `SmSE`, *has a* finite precision protocol *if when* $\mathcal{M}$ *enters the query state with the word* z *in the query tape*, *the* `SmSE` *carries out the experiment with the cannon in the position* $z' \in [z - \varepsilon, z + \varepsilon]$, *for a fixed* $\varepsilon > 0$. *The probability distribution of* z' *in this interval is considered to be uniform and independent of previous experiments.*

We can now formally define the `SmSM`:

Definition 3 (Smooth Scatter Machine). A Smooth Scatter Machine, or `SmSM` is an `ADTM` using a `SmSE` as a physical oracle, with a specified time schedule T, a fixed vertex position $y \in [0, 1]$ and equipped with one of the communication protocols above.

We now define that for all `SmSM`'s, regardless of the protocol, when the associated `SmSE` carries out the experiment with query word z we have that:

(1) If the particle crosses the upper sensor in $T(|z|)$ time steps or less, then the `SmSM` resumes the computation in the output state q_r;
(2) If the particle crosses the lower sensor in $T(|z|)$ time steps or less, then the `SmSM` resumes the computation in the output state q_ℓ;
(3) If the experiment times out, then the `SmSM` resumes the computation in the output state q_t.

We define decidability by an `SmSM`, noting that in the case of error prone `SmSM` every query to the `SmSE` is a probabilistic event, hence the acceptance criteria for these machines is similar to that of probabilistic Turing machines.

Definition 4. Let $A \subseteq \Sigma^\star$ be a set of words over the alphabet Σ. We say that a `SmSM` $\mathcal{M}$ with the infinite precision protocol decides A if for every $w \in \Sigma^\star$, if $w \in A$, $\mathcal{M}$ accepts w, and if $w \notin A$, $\mathcal{M}$ rejects w. We say that $\mathcal{M}$ decides A in *polynomial* time if there exists a polynomial p such that for every $w \in \Sigma^\star$, if $w \in A$, $\mathcal{M}$ accepts w in $p(|w|)$ or less steps, and if $w \notin A$, $\mathcal{M}$ rejects w in $p(|w|)$ or less steps.

Definition 5. Let $A \subseteq \Sigma^\star$ be a set of words over the alphabet Σ. We say that a SmSM $\mathcal{M}$ with an error-prone protocol decides A if there exists a $\gamma < \frac{1}{2}$ such that for every $w \in \Sigma^\star$, if $w \in A$, $\mathcal{M}$ accepts w with error probability less than γ, and if $w \notin A$, $\mathcal{M}$ rejects w with error probability less than γ. We say that $\mathcal{M}$ decides A in *polynomial* time if there exists a polynomial p such that for every $w \in \Sigma^\star$, if $w \in A$, $\mathcal{M}$ accepts w in $p(|w|)$ or less steps with error probability less than γ, and if $w \notin A$, $\mathcal{M}$ rejects w in $p(|w|)$ or less steps with error probability less than γ.

We will only be considering in this paper the infinite precision case.

2.3.1. *The linear search algorithm*

It is important to understand how we can perform measurements using the SmSM. We consider $x\rfloor_\ell$ to be the prefix of size ℓ of x, if $|x| \geq \ell$, or the word x padded with a number k of $0's$ such that the resulting word $x0^k$ has size ℓ, otherwise. In Algorithm 1, we specify the linear search algorithm for infinite precision SmSM's.

This algorithm allow us to measure the vertex position y bit by bit, assuming there are no timeouts. If there is a timeout for the cannon position $z\rfloor_\ell$, then this means that $t_{exp}(z\rfloor_\ell) > T(\ell)$, and since from Eq. (1) we have that $t_{exp}(z\rfloor_\ell) \leq \frac{B}{|z\rfloor_\ell - y|}$ for some B, we would have that $|z\rfloor_\ell - y| < \frac{B}{T(\ell)}$.

If there are no timeouts, then we need ℓ runs to the experiment each taking time at most $T(\ell)$, so we need $\mathcal{O}(\ell T(\ell))$ time steps. If, on the other hand, there is a timeout, then the algorithm needs to run the cycle at most ℓ times, hence we obtain the same complexity.

Algorithm 1 Linear search algorithm for infinite precision SmSM

```
input ← $\ell$ is the desired precision
$x_0 \leftarrow 0$
$x_1 \leftarrow 1$
$z \leftarrow 0$
while $x_1 - x_0 > 2^{-\ell}$ do
    $z \leftarrow (x_0 + x_1)/2$
    $s \leftarrow$ state after executing SmSE with cannon in position $z\rfloor_\ell$
    if $s = q_r$ then
        $x_0 \leftarrow z$                                  ▷ we know that $z\rfloor_\ell > y$
    else if $s = q_\ell$ then
        $x_1 \leftarrow z$                                  ▷ we know that $z\rfloor_\ell < y$
    else
        return "timeout"
    end if
end while
return $z\rfloor_l$
```

2.3.2. *The Cantor set $\mathcal{C}_3$*

The Cantor set of base 3 ($\mathcal{C}_3$) is the set of real numbers of the form $x = \sum_{k=1}^{\infty} x_k 2^{-3k}$, where $x_k \in \{1, 2, 4\}$. In other words, $\mathcal{C}_3$ is the set of numbers composed by the concatenation of triples of the form 100, 010 or 001. The following proposition regarding these numbers will be useful to prove the lower bounds of the computational power of the infinite precision SmSM.

Proposition 6 (Beggs, Costa, Poças, and Tucker in [2, 7]). *For every $x \in \mathcal{C}_3$ and for every dyadic rational $z \in [0, 1]$ with size $|z| = m$, if $|x - z| \leq 2^{-(i+5)}$, then x and z coincide in the first i bits, and $|x - z| > 2^{-(m+10)}$.*

Proof. Suppose that x and z coincide in the first $i - 1$ bits but differ in the ith. We want to show that $|x - z| > 2^{-(i+5)}$. We have two cases to consider:

(1) $z < x$. In this case the ith bits of z and x are 0 and 1 respectively. In the worst case the binary expansion of z is followed by 1's after the ith bit. Because $x \in \mathcal{C}_3$, following the ith bit there can be no more than four consecutive bits with the value 0, hence in the worst case, the binary expansions of z and x from the ith bit onwards are respectively $011111\cdots$ and $100001\cdots$ and so $|x - z| > 2^{-(i+5)}$.
(2) $z > x$. In this case the ith bits of z and x are 1 and 0 respectively. In the worst case, the binary expansion of z is followed by 0's after the ith bit. Because $x \in \mathcal{C}_3$, following the ith bit there can be no more than two consecutive bits with the value 1, hence in the worst case, the binary expansions of z and x from the ith bit onwards are respectively $1000\cdots$ and $0110\cdots$ and so $|x - z| > 2^{-(i+3)}$.

So we have established that if $|x - z| \leq 2^{-(i+5)}$, then x and z coincide in the first i bits. Now we need to prove that for any dyadic rational z with $|z| = m$, $|x - z| > 2^{-(m+10)}$. If $z \neq x\rfloor_m$ this is straightforward. Suppose now that $z = x\rfloor_m$. Since $x \in \mathcal{C}_3$, it can have at most four consecutive bits with the value 0, hence, since all the bits of the binary expansion of z from the mth bit onwards are 0, we have that z and x's binary expansion coincide at most in $m + 4$ bits, and differ in the $(m + 5)$th. Therefore, by the first part of the proof of this proposition, $|x - z| > 2^{-(m+10)}$. □

We use numbers in Cantor set to encode words in the following way: take $w \in \{0, 1\}^\star$, $c(w)$ is a codification of w that consists in substituting every 1 by 010 and every 0 by 100. Furthermore, considering a prefix advice function $f : \mathbb{N} \to \{0, 1\}^\star$,[c] we denote its encoding as the real number given by $y(f) = \lim y(f)(n)$, where

[c] An advice function $f : \mathbb{N} \to \{0, 1\}^\star$ is a total function which assigns a word for each (input size) $n \in \mathbb{N}$. A prefix advice function is an advice function f such that, for any $n, m \in \mathbb{N}$, if $n < m$, then $f(n)$ is a prefix of $f(m)$. If $\mathcal{F}$ is a class of advice functions, then we denote by $\mathcal{F}\star$ the class of prefix functions in $\mathcal{F}$.

$y(f)(n)$ is defined recursively as follows (take $f(n+1) = f(n).s$):

$$y(f)(0) = 0.c(f(0))$$

$$y(f)(n+1) = \begin{cases} y(f)(n).c(s) & \text{if } n+1 \text{ is not a power of 2} \\ y(f)(n).c(s).001 & \text{if } n+1 \text{ is a power of 2.} \end{cases} \tag{2}$$

3. Lower and Upper Bounds

3.1. *Lower bounds*

We will now provide a lower bound for the computational power of infinite precision SmSM's. We begin by proving an auxiliary proposition.

Proposition 7. *Take a prefix function $f \in$ log.*[d] *There exists an SmSM $\mathcal{M}$ clocked in polynomial time in n with time schedule $T(k) \in \Theta(2^k)$ and access to a SmSE with wedge in the position $y(f)$ that determines $f(n)$.*

Proof. Suppose that $2^{m-1} < n \leq 2^m$ for positive m. Since $f \in$ log, $|c(f(n))| \leq a\lceil \log n \rceil + b$ for some $a, b \in \mathbb{N}$, which means, according to Eq. (2), that we need at most $k = (a+3)m + b$ bits of $y(f)$ to obtain $f(n)$. Note that k is logarithmic in n. We now apply the linear search algorithm introduced in Sec. 2.3.1. Firstly, we need to guarantee that there are no timeouts during its execution. By Proposition 6, since $y(f) \in \mathcal{C}_3$, any dyadic rational x with $|x| = k$ is such that $|x - y(f)| > 2^{-(k+10)}$. Using this fact and Eq. (1) for the experimental time, we conclude that $t_{exp}(x) < B.2^{k+10}$, and so if we choose our time schedule to be $T(k) = B.2^{k+10}$ we guarantee that there will be no timeouts during the execution of the linear search algorithm.

As seen in Sec. 2.3.1, the linear search algorithm will take time $\mathcal{O}(kT(k))$ which, since k is logarithmic in n, is $\mathcal{O}(n \log n)$ which is polynomial in n. Finally, given $y(f)(2^m)$, we can obtain $f(2^m)$ in polynomial time in n. □

Proposition 8. *If $A \in$ P$/\log\star$, then there exists a SmSM $\mathcal{M}$ clocked in polynomial time with infinite precision and time schedule $T(k) \in \Theta(2^k)$ that decides A.*

Proof. $A \in$ P$/\log\star$, therefore there exists a set $D \in$ P, witnessed by some Turing machine $\mathcal{M}'$, and a prefix function $f \in$ log such that for all $w \in \{0,1\}^\star$, and for all $n \geq |w|$, $w \in A$ if and only if $\langle w, f(n) \rangle \in D$. We outline in Algorithm 2 the SmSM $\mathcal{M}$ with vertex position of the associated SmSE at $y(f)$, as described in Eq. (2).

According to the proof of Proposition 7, if we choose the schedule $T(k) = B.2^{k+10}$ for $\mathcal{M}$, and using the linear search algorithm, we can conclude step 3 in polynomial time in 2^m (and hence in polynomial time in $|w|$). Moreover, since $f \in$ log, we have that $|f(2^m)| \in \mathcal{O}(m)$, and so $|\langle w, f(2^m) \rangle| \in \mathcal{O}(|w| \log |w|) \subseteq \mathcal{O}(|w|^2)$. Thus we can conclude that steps 4 and 5 take polynomial time in $|w|$; hence $\mathcal{M}$ decides A in polynomial time. □

[d] log denotes the class of advice functions such that $|f(n)| \in \mathcal{O}(\log(n))$.

Algorithm 2 SmSM $\mathcal{M}$

```
1: input ← $w$
2: find $m$ such that $2^{m-1} < |w| \leq 2^m$
3: use the SmSE to obtain $f(2^m)$        ▷ polynomial in $2^m$ time (Proposition 7)
4: build $\langle w, f(2^m) \rangle$
5: simulate $\mathcal{M}'$ on $\langle w, f(2^m) \rangle$
6: if $\mathcal{M}'$ accepts $\langle w, f(2^m) \rangle$ then
7:     accept $w$;
8: else
9:     reject $w$
10: end if
```

3.2. *Boundary numbers*

Boundary numbers are the closest values to the vertex position y for which we can fire the cannon and get a result before timing out. Since the wedge curve is symmetric, we expect that we will have one of these numbers greater than y and another smaller. They are defined as follows:

Definition 9. Let y be the vertex position and T the time schedule for a SmSM. For a fixed query size k, we define the boundary numbers[e] as $0 < \ell_k, r_k < 1$, such that $\ell_k < y < r_k$ and $t_{exp}(\ell_k) = t_{exp}(r_k) = T(k)$.

Given an experiment with cannon position z with $|z| = k$, we have three possible cases:

(1) $l_k < z < r_k$. Here, since z is closer to the vertex position y than the boundary numbers, we conclude that $t_{exp}(z) > t_{exp}(\ell_k)$ and $t_{exp}(z) > t_{exp}(r_k)$. Since $t_{exp}(\ell_k) = t_{exp}(r_k) = T(z)$, the experiment will timeout.
(2) $z \leq \ell_k$. We have that $z \leq \ell_k < y$, so $t_{exp}(z) \leq t_{exp}(\ell_k)$ because $z < y$ and z is further away from the vertex position than ℓ_k. Therefore, from Definition 9, $t_{exp}(z) \leq T(k)$, and so the experiment will not timeout and the particle will be detected in the left collecting box.
(3) $z \geq r_k$. Very similar to the previous case, the experiment will not timeout and the particle will be detected in the right collecting box.

To help visualize how these values behave, it is helpful to note that as k increases, ℓ_k and r_k get closer to the vertex position y (see Fig. 2). This is because the time schedule T is an increasing function, and t_{exp} is strictly increasing in $[0, y[$ and strictly decreasing in $]y, 1]$.

[e]We will always consider the binary expansion of these numbers with no infinite sequence of 1's.

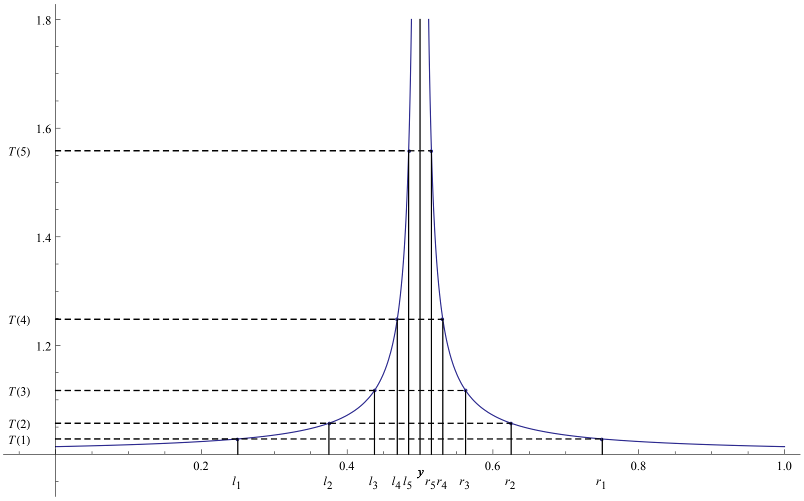

Fig. 2. Behavior of ℓ_k and r_k.

These values, ℓ_k and r_k, are useful because if we have access to their prefixes we can simulate queries to the SmSE.

Proposition 10. *Take a SmSM with left boundary number ℓ_k, for some fixed k. If we have access to $\ell_k\downharpoonleft_k$ then, for any query z with $|z| = k$, we can determine in linear time in k whether the SmSM will make a transition to state q_ℓ with query z.*

Proof. Take query z with $|z| = k$, and $\ell_k\downharpoonleft_k$. Since $|z| = |\ell_k\downharpoonleft_k| = k$, the comparison of these words will be done in time $\mathcal{O}(k)$. Suppose that $z > \ell_k\downharpoonleft_k$. Then $z > \ell_k$ and therefore the outcome of the SmSE with z will either be the right collecting box or timeout, hence the SmSM will not resume the computation in state q_ℓ. Suppose that $z \leq \ell_k\downharpoonleft_k$. Since $\ell_k\downharpoonleft_k \leq \ell_k$, we have that $z \leq \ell_k$. We know that the SmSM will resume in state q_ℓ. □

For the boundary number r_k we cannot prove a result as straightforward as Proposition 10. This is because if we have a query z of size k, and $z = r_k\downharpoonleft_k$ we cannot infer if $z = r_k$ or $z < r_k$. Due to this, we would not be able to determine the result of executing the SmSE with query z. However, if we know some additional information about r_k (namely if it has a non-zero bit in some position k' with $k' > k$) this becomes feasible.

Definition 11. Given a fixed $k \in \mathbb{N}$, and a SmSM with r_k, right boundary number of order k, we definine σ_k as:

$$\sigma_k = \begin{cases} 1 & r_k\downharpoonleft_k = r_k \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Therefore we have that $\sigma_k = 1$ if and only if r_k can be expressed as a dyadic rational of size k.

Proposition 12. *Take a* SmSM *with right boundary number* r_k*, for some fixed* k*. If we have access to* $r_k\downharpoonleft_k$ *and* σ_k *then, for any query* z *with* $|z| = k$*, we can determine if the* SmSM *will make a transition to state* q_r *with query* z*, in linear time over* k*.*

Proof. Take query z with $|z| = k$, $r_k\downharpoonleft_k$ and σ_k. Since $|z| = |r_k\downharpoonleft_k| = k$, the comparison of these words can be done in time $\mathcal{O}(k)$. Suppose that $z = r_k\downharpoonleft_k$. We have two cases, either $\sigma_k = 1$ or $\sigma_k = 0$. In the former case, by definition of σ_k we have that $r_k = r_k\downharpoonleft_k$ and so $z = r_k$. The result will be state q_r. If, on the other hand, $\sigma_k = 0$, again by definition of σ_k we have that $r_k > r_k\downharpoonleft_k$; therefore $r_k > z$ and thus the outcome of the experiment with cannon position z will either be q_ℓ or q_t.

The cases where $z < r_k\downharpoonleft_k$ and $z > r_k\downharpoonleft_k$ are handled very similarly to what was done in the proof of Proposition 10. □

Unifying Propositions 10 and 12, the following proposition becomes trivial.

Proposition 13. *Take a* SmSM *with boundary numbers* ℓ_k *and* r_k*, for some fixed* k*. If we have access to* $\ell_k\downharpoonleft_k$*,* $r_k\downharpoonleft_k$*, and* σ_k *then, for any query* z *with* $|z| = k$*, we can determine in linear time over* k *to which state the* SmSM *will make a transition to, after executing the* SmSE *with query* z*.*

3.3. *Upper bounds*

We fix now the upper bounds for the SmSM clocked in polynomial time.

Proposition 14 (Ambaram, Beggs, Costa, Poças, and Tucker in [2]). *If* $A \subseteq \{0, 1\}^\star$ *is decided by an infinite precision* SmSM *clocked in polynomial time with time schedule* $T(n) \in \Omega(2^n)$*, then* $A \in P/\log^2\star$*.*

Proof. Let $\mathcal{M}$ be the infinite precision SmSM that decides A in polynomial time $\mathcal{O}(n^k)$, and T be the time schedule, such that $T(n) \in \Omega(2^n)$. Since the schedule is exponential, queries to the SmSE can only be at most logarithmic in size, otherwise an experiment would take more than polynomial time. Let $\lambda = a\lfloor\log n\rfloor + b$ be the bound for query sizes of $\mathcal{M}$. Take the following advice function f, such that $f(\lambda)$ encodes the prefixes of ℓ_k and r_k needed to simulate queries to the SmSE of size up to λ:

$$f(\lambda) = \ell_1\downharpoonleft_1\#r_1\downharpoonleft_1\#\sigma_1\#\ell_2\downharpoonleft_2\#r_2\downharpoonleft_2\#\sigma_2\#\ldots\ell_\lambda\downharpoonleft_\lambda\#r_\lambda\downharpoonleft_\lambda\#\sigma_\lambda\#.$$

It is easy to see that $|f(\lambda)| = \left(4\lambda + 2\sum_{i=1}^{\lambda} i\right)$, and hence $|f(\lambda)| \in \mathcal{O}(\lambda^2)$, i.e. $|f(\lambda)| \in \mathcal{O}(\log^2(n))$. Take a Turing machine $\mathcal{M}'$, receiving input $\langle w, f(n')\rangle$, with $|w| = n$ and $n' \geq n$ (where n' is the least power of 2 greater or equal to n). Suppose $\mathcal{M}'$ mimics $\mathcal{M}$, replacing queries z to the SmSE by fetching $\ell\downharpoonleft_{|z|}$, $r\downharpoonleft_{|z|}$, and $\sigma_{|z|}$ from $f(n')$, and simulating the experiment. This simulation is done in time

$\mathcal{O}(\lambda) = \mathcal{O}(\log n)$ according to Proposition 13. Obtaining the boundary numbers from $f(n')$ is done in time $\mathcal{O}(\log^2 n)$ where n is the size of the input. Thus, we can decide A in polynomial time with the prefix advice function $f \in \log^2$. □

If we use a time schedule T such that $T(k) \in \Omega(2^k)$, it is possible to codify the relevant information relative to the boundary numbers in an advice function $f \in \texttt{log}$.

Proposition 15 (Beggs, Costa, Poças, and Tucker in [2, 7]). *Given the boundary numbers for a SmSM with time constructible schedule $T(k) \in \Omega(2^k)$ it is possible to define a prefix advice function f such that $f(\lambda)$ encodes all the prefixes of the boundary numbers with size up to λ and $|f(\lambda)| \in \mathcal{O}(\lambda)$.*

Proof. Take a SmSM with vertex position y and time schedule $T(k) \in \Omega(2^k)$. We know that $\exists k_0, \alpha \in \mathbb{N}$ such that $\forall k \geq k_0 (T(k) \geq \alpha 2^k)$. From now on we fix a $k \geq k_0$. By the definition of r_k, $t_{exp}(r_k) = T(k)$, and using the lower bound on experimental time provided by Eq. (1), we conclude that $\exists c(|y - r_k| < 2^{c-k})$, and so, since by definition of r_k, $y < r_k$:

$$y < r_k < y + 2^{c-k}. \tag{4}$$

Now take $r_k\rfloor_k = v_k w_k$, with $|v_k| = k - c$ and $|w_k| = c$. It is easy to see that $v_k \leq r_k$ and $r_k < v_k + 2^{k-c}$, and thus, using Eq. (4), we get $y - 2^{c-k} < r_k - 2^{c-k} < v_k \leq r_k < y + 2^{c-k}$. We conclude that

$$|v_k - y| < 2^{c-k}. \tag{5}$$

The word v_k can be of one of the three following forms, for some λ:

(1) $v_k = \ldots 10^\lambda$. From Eq. (5) we conclude that either $y\rfloor_{k-c} = \ldots 10^\lambda$ or $y\rfloor_{k-c} = \ldots 01^\lambda$. Therefore, since we have $|v_{k+1} - y| < 2^{c-k-1}$ (again from Eq. (5)), v_{k+1} can be of one of the following forms:

$$\ldots 10^{\lambda+1} \qquad \ldots 10^\lambda 1 \qquad \ldots 01^{\lambda+1} \qquad \ldots 01^\lambda 0;$$

(2) $v_k = \ldots 01^\lambda$. With the same reasoning as the case above, we conclude that either $y\rfloor_{k-c} = \ldots 01^\lambda$ or $y\rfloor_{k-c} = \ldots 01^{\lambda-1}0$, and thus v_{k+1} can be of one of the following forms:

$$\ldots 01^\lambda 0 \qquad \ldots 01^{\lambda+1} \qquad \ldots 01^{\lambda-1}00 \qquad \ldots 01^{\lambda-1}01;$$

(3) $v_k = 0^{k-c}$. This case is even simpler because $y\rfloor_{k-c} = 0^{k-c}$ and so we are left with just two possible cases for v_{k+1}, 0^{k-c+1} and $0^{k-c}1$.

Thus, for $k \geq k_0$, if we know v_k, we have at most four possible cases for v_{k+1}, so just two bits of information are needed to compute it. We will denote these bits as b_{k0} and b_{k1}. Furthermore, if we have v_{k+1} we just need w_{k+1} (that is, c additional bits) to compute r_{k+1}. Hence, for all $k \geq k_0$, we just need r_k and $(c + 2)$ bits of information to compute r_{k+1}. The same conclusion can be drawn for the lower boundary numbers ℓ_k, using an analogous procedure, splitting these numbers as $\ell_k\rfloor_k = x_k y_k$, with $|u_k| = k - c$ and $|y_k| = c$, where c is such that $|y - r_k| < 2^{c-k}$ and $|y - \ell_k| < 2^{c-k}$. We will denote the two bits of information that we need to compute x_{k+1} from x_k by b_{k2} and b_{k3}. Thus the required advice function can be given by:

$$f(n) = \begin{cases} \ell_1\rfloor_1\#r_1\rfloor_1\#\sigma_1\#\ell_2\rfloor_2\#r_2\rfloor_2\#\sigma_2\#\ldots\ell_n\rfloor_n\#r_n\rfloor_n\#\sigma_n\# & n < k_0 \\ f(k_0-1)\#\#x_{k_0}\#y_{k_0}\#v_{k_0}\#w_{k_0}\#\sigma_{k_0} & n = k_0 \\ f(n-1)\#b_{n2}b_{n3}\#y_n\#b_{n0}b_{n1}\#w_n\#\sigma_n & n > k_0. \end{cases} \tag{6}$$

Since w_n and y_n have constant size c for all $n > k_0$, $|f(n)|$ will be asymptotically linear. □

Following the proof of Proposition 14, but using function (6) instead, constructed in the proof of Proposition 15, we are able to reduce the upper bound of infinite precision, polynomially clocked SmSM's, for time schedules in $\Omega(2^k)$.

Proposition 16. *If A is decided by a SmSM clocked in polynomial time with infinite precision and exponential protocol $T(k) \in \Omega(2^k)$, then $A \in P/\log\star$.*

With Propositions 16 and 8, we immediately have the following Proposition:

Proposition 17. *A is decided by a SmSM clocked in polynomial time with infinite precision and exponential time schedule $T(k) \in \Omega(2^k)$ if and only if $A \in P/\log\star$.*

As it was noted in [2], it is an open problem to know if this proposition holds if we consider that the time schedules considered are not in $\Omega\left(2^k\right)$.

4. Impact of Time Schedules

We will now focus on the study of the impact of time schedules on the computational power of the SmSM clocked in polynomial time. The results reported in this section complete the research undergone in [8], providing full proofs of the propositions left unproved. For this purpose, we fix the position of the vertex of the SmSE to $y = 1/2$ and consider different classes of time schedules. Note that the vertex position could be fixed at any computable value.

Definition 18. We denote by AP(f) the class of sets decidable by some SmSM clocked in polynomial time, using the time schedule $f : \mathbb{N} \to \mathbb{N}$, where f is an increasing total function, infinite precision and vertex position $y = 1/2$ for the associated SmSE. If $\mathcal{F}$ is a class of increasing total functions then AP $(\mathcal{F}) = \bigcup_{f\in\mathcal{F}}$ AP (f).

4.1. *Fixing the curve*

As we will see below, the class AP $(\mathcal{F})$, for some set $\mathcal{F}$ of functions, depends on the experimental time. In Propositions 20 and 22 we will be considering the following equation for the experimental time, since we are merely interested in how fast t_{exp}

grows as z approaches y:

$$t_{exp}(z) = \frac{1}{|z-y|}. \tag{7}$$

Furthermore, since $t_{exp}(r_k) = t_{exp}(\ell_k) = T(k)$ and $y = 1/2$, then

$$\ell_k = \frac{1}{2} - \frac{1}{T(k)} \quad \text{and} \quad r_k = \frac{1}{2} + \frac{1}{T(k)}. \tag{8}$$

It is important to notice that although $y = 1/2$ is computable, the non-uniform bounds of the computational power of the SmSM so far identified as $\texttt{P/poly}$, $\texttt{P}/\log\star$, and $\texttt{P}/\log^2\star$ do not collapse to P, because we can have non-computable time schedules and so the boundary numbers still might not be computable implying that the corresponding advice functions can also be non-computable.

Assuming we have an explicit formula for the experimental time, and furthermore that we have $T(k)$, then it is apparent that we can compute $\ell_k\rfloor_k$, $r_k\rfloor_k$, and σ_k. It is desirable not to impose this explicit equation on the experimental time in order to prove more general versions of Propositions 22 and 20. There are therefore two approaches to deal with these boundary values, one is where we have explicit access to this time, and thus are able to compute ℓ_k and r_k; another approach consists in the case where the experimental time is implicit, in this case ℓ_k and r_k are not necessarily computable, however prefixes of these values can be used to obtain the upper bounds, which is what we did in Proposition 19.

4.2. *Boosting* $\texttt{P}/\log\star$

Let IN denote the class of increasing total functions, CI the class of computable increasing total functions, and TC the class of all strictly increasing time constructible functions. Since $\texttt{TC} \subset \texttt{CI} \subset \texttt{IN}$ we can conclude that $\texttt{AP}(\texttt{TC}) \subseteq \texttt{AP}(\texttt{CI}) \subseteq \texttt{AP}(\texttt{IN})$. Note that for time schedules in CI or IN the SmSM may not count the time it should wait for calls over the physical oracle internally, because time schedules in these conditions are not time constructible in general. Nevertheless the SmSM proceed in busy waiting until the end of the call. In this section we describe these three classes in terms of non-uniform complexity. In particular, in Proposition 23 we show that any recursive set decidable by an oracle Turing machine running in polynomial time by means of a tally[f] oracle T, possibly non-decidable, can also be decidable by another oracle Turing machine using some decidable tally oracle T'.

Proposition 19. *$\texttt{AP}(\texttt{IN}) = \texttt{P/poly}$.*

Proof. $\supseteq$. Let $A \in \texttt{P/poly}$. Polynomial advice Turing machines are polynomial time equivalent to Turing machines with tally sets as oracles (see [3]). Hence there

[f] A tally set, also called unary language, is a subset of $\{0^k : k \in \mathbb{N}\}$. In other words, in a tally set all words are comprised of 0's.

exists a tally set D such that $A \in \mathtt{P}^D$. Let $\mathcal{M}$ be the Turing machine clocked in polynomial time p that witnesses this fact.

Consider the following time schedule:

$$T(k) = \begin{cases} 2k+1 & \text{if } 0^k \in D \\ 2k & \text{if } 0^k \notin D. \end{cases} \tag{9}$$

We can consider that this time schedule is originated by simulating a clock for $2k+1$. This simulation has disturbances induced by oracle D, forcing it to skip one transition if $0^k \notin D$, thus producing a procedure that effectively carries out $T(k)$ transitions and then halts. In addition, consider a sequence of words z_k such that $|z_k| = k$ and $2k < t_{exp}(z_k) < 2k+1$. If such a sequence exists and we can compute z_k in polynomial time, then we can simulate calls to the oracle D using an infinite precision SmSM with access to a SmSE with vertex position at $y = 1/2$, cannon position at z_k, and time schedule T given by Eq. (9), in the way specified by Algorithm 3.[g]

The time to execute step 3 in the algorithm above is linear in k because the time schedule T is also linear in k. Hence, step 2 aside, this simulation takes polynomial time in k, with k being polynomial in the size of the input of $\mathcal{M}$.

We now show how to compute these numbers z_k. Given k, the sequence z_k is such that $2k < t_{exp}(z_k) < 2k+1$. Using the bounds on the experimental time given by Eq. (7) we conclude that:

$$2k < \frac{1}{|z_k - 1/2|} \quad \text{and} \quad \frac{1}{|z_k - 1/2|} < 2k+1. \tag{10}$$

Choosing $z_k > 1/2$, we get that $\frac{1}{2} + \frac{1}{2k+1} < z_k < \frac{1}{2} + \frac{1}{2k}$. If we choose large enough values for k, then the difference between the lower and upper bounds of z_k is greater than 2.2^{-k}. Thus, if we choose $z_k = (\frac{1}{2} + \frac{1}{2k})\rfloor_k - 2^{-k}$, we guarantee that $2k < t_{exp}(z_k) < 2k+1$, with $|z_k| = k$. We can, without loss of generality, admit that there are only large enough elements in the tally set D.

$\subseteq$. We have that $A \in \mathtt{AP(IN)}$. Therefore, by Definition 18, there exists an infinite precision SmSM $\mathcal{M}$ with vertex position $y = 1/2$ and time schedule $T \in \mathtt{IN}$ that

Algorithm 3 Simulating a call to tally oracle D

```
1: input ← $0^k$
2: compute $z_k$
3: run the SmSE with cannon at $z_k$; let $s$ be the resulting state of the SmSM
4: if $s = q_t$ then resume the computation in state NO
5: else resume the computation in state YES
6: end if
```

[g]Algorithm 3, line 4: timeout, hence $T(k) = 2k$ and so $0^k \notin D$. Algorithm 3, line 5: no timeout, hence $T(k) = 2k+1$ and so $0^k \in D$.

decides A when clocked in polynomial time p. Consider an input w with $|w| = n$. On a computation of $\mathcal{M}$ on w, there can be at most $p(n)$ calls to the SmSE. Hence, if we are able to simulate these experiment calls in polynomial time, with a polynomial size advice, then $A \in \texttt{P/poly}$.

The size of queries of $\mathcal{M}$ to the SmSE is bounded by $p(n)$. Take the advice function f, similar to the one defined in the proof of Proposition 14:

$$f(n) = \ell_1\lfloor_1\#r_1\lfloor_1\#\sigma_1\#\ell_2\lfloor_2\#r_2\lfloor_2\#\sigma_2\#\cdots\#\ell_{p(n)}\lfloor_{p(n)}\#r_{p(n)}\lfloor_{p(n)}\#\sigma_{p(n)}\#. \tag{11}$$

We have that $f \in \texttt{poly}$, because $|f(n)| \in \mathcal{O}(p^2(n))$. Additionally, given a query of size $k < p(n)$, we can extract $\ell_k\lfloor_k$, $r_k\lfloor_k$, and σ_k from $f(n)$ in polynomial time in n, and simulate the experiment in linear time in k, from Proposition 13. Therefore, $A \in \texttt{P/poly}$. □

Regarding the first part of the former proof, it is clear that $\lim_{k\to\infty} z_k = 1/2$. This is true since $\frac{1}{2} + \frac{1}{2k+1} < z_k < \frac{1}{2} + \frac{1}{2k}$, and we have that $\lim_{k\to\infty} \frac{1}{2k+1} = \lim_{k\to\infty} \frac{1}{2k} = 0$. But it is not at all intuitive how there can be a sequence of cannon positions z_k (of size k) that produce a sequence of linearly increasing values of $t_{exp}(z_k)$, since t_{exp} increases exponentially in the number of bits of precision of the queries. The only explanation is that z_k converges very slowly to $1/2$. Figure 3 helps to illustrate this fact, by showing the increase needed in k in order for z_k to obtain a new bit of $1/2$, which appears to be exponential in the desired precision. For example, $z_{10\,000}$ only coincides in its first 14 bits with $1/2$.

The proposition above is a more general result than the one presented in [8], because we removed the explicit experimental time (Eq. (7)) constraint.

It should be apparent from Proposition 19 that, even though we removed all possible non-computability from the vertex position of the experiment, considering non-computable time schedules allows to surpass the power of conventional infinite precision SmSM's, when clocked in polynomial time. Furthermore, when the schedule is time constructible as in Proposition 20, all the sources of non-computability are

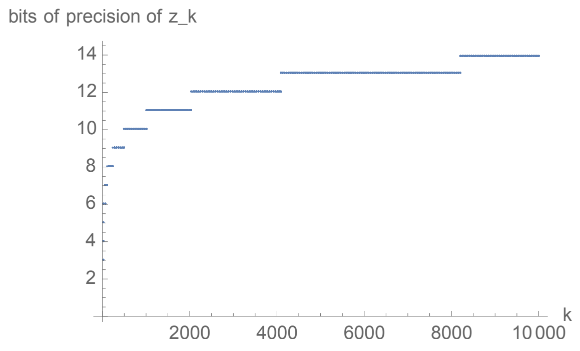

Fig. 3. Precision of z_k.

stripped from the SmSM, which is why the power of these machines, when clocked in polynomial time, drops to P.

Proposition 20. *AP (TC) = P.*

Proof. Suppose $A \in$ P, then A is decidable in polynomial time by a deterministic oracle Turing machine on help by the empty oracle. Thus $A \in$ AP (TC) since no calls are made.

If $A \in$ AP (TC), then A is decidable by an analogue-digital machine $\mathcal{M}$ in polynomial time using the physical oracle with vertex at position $y = 1/2$, working with infinite precision, and an increasing time constructible function T as a time schedule. We devise a deterministic machine $\mathcal{M}'$ that, given an input of size n, behaves like $\mathcal{M}$ and simulates any oracle query of size k in polynomial time. We notice that given a query word of size k, it is possible to compute $T(k)$ in polynomial time since T is time constructible by assumption and there exists a polynomial $p(n)$ bounding the running time of $\mathcal{M}$, so we have $T(k) < p(n)$. Then, it computes $\ell_k\rfloor_k$, $r_k\rfloor_k$ where ℓ_k, r_k are the boundary numbers from Eq. (8). Finally, it compares these value with the query word z of size k. The computation of $\ell_k\rfloor_k$, $r_k\rfloor_k$ and the comparison of z with these numbers can be done in polynomial time, so $\mathcal{M}'$ also decides A in polynomial time. It follows that $A \in$ P. □

Proposition 21. *If $A \in$ P^D for some tally decidable set D, then $A \in$ AP (CI).*

Proof. Since $A \in$ P^D and D is tally, then $A \in$ P/poly. Consider the same SmSM as in the proof of Proposition 19, with time schedule T defined in Eq. (9). Since $A \in$ P/poly, this SmSM decides A in polynomial time. Furthermore, T is a computable function, because $D \in$ REC. Therefore $A \in$ AP (CI). □

Proposition 22. *AP (CI) $\subseteq$ P/poly $\cap$ REC.*

Proof. Let $A \in$ AP (CI). Since AP (CI) $\subseteq$ AP (IN) and AP (IN) = P/poly, we conclude that $A \in$ P/poly. We now show that $A \in$ REC. Consider the SmSM machine $\mathcal{M}$ and the computable time schedule T that witnesses $A \in$ AP (CI). Similarly to the previous proposition, we devise a deterministic Turing machine $\mathcal{M}'$ such that, for any $k \in \mathbb{N}$ it computes $\ell_k\rfloor_k$, $r_k\rfloor_k$ where ℓ_k, r_k are the boundary numbers from Eq. (8). To decide A, $\mathcal{M}'$ just has to simulate $\mathcal{M}$ and, using the fact that both $T(k)$ and t_{exp} are computable, whenever $\mathcal{M}$ reaches a query state, it computes $\ell_k\rfloor_k$, $r_k\rfloor_k$ and compares the result with the query word. It follows that $A \in$ REC. □

The converse of Proposition 22 is not straightforward since, if a set A is decidable by an oracle Turing machine clocked in polynomial time using an arbitrary tally set T as oracle, then we cannot assume at once that T can be replaced by a recursive tally set. To solve this open problem left on [8], and complete the proof of the lower bound of AP (CI) (see Proposition 24), we will show now that given some recursive set A decidable by a deterministic oracle Turing machine bounded in polynomial time using a non-recursive tally set as oracle, it is possible to obtain a recursive tally set for which there exists another deterministic oracle Turing machine clocked in polynomial time with that recursive oracle that can decide the same set A.

To prove the statement, first we need to order finite tally sets. More specifically, we want to list them so that given some finite tally set we can perform a test on all of its subsets. Let $T = \{0^{\tau_1}, 0^{\tau_2}, \dots\}$ be a tally set with $\tau_m < \tau_n$ whenever $m < n$. We can order the finite subsets of T using the following enumeration e:

- $e(\emptyset) = 0$; $e(\varepsilon) = 1$;
- $e(\{0^{k_1}, 0^{k_2}, \dots, 0^{k_n}\}) = \sum_{i=1}^{n} 2^{k_i}$.

It easy to show that this enumeration is actually a bijection between $\mathbb{N}$ and the class of finite tally sets since we can simply represent these sets as a list of the form $\{k_1, \dots, k_n\}$. Therefore, if we want to perform some test on all subsets of some tally T up to a fixed word size n, we compute $e^{-1}(0)$, $e^{-1}(1)$, $e^{-1}(2), \dots, e^{-1}(2^n)$ disregarding sets that are not subsets of T. The enumeration is also monotonic in the sense that if $A \subset B$ then $e(A) < e(B)$. In particular, if T is itself finite, then ordering its subsets can be done in finite time as we only need to compute at most $e(T)$ subsets. The overall complexity of ordering the subsets of T is not relevant as we only intend to use this ordering in the proof of Proposition 23 to show that some tally set is recursive. Let $S_i = e^{-1}(i)$ denote the tally set encoded by $i \in \mathbb{N}$.

Proposition 23. *If $A \in$ P/poly $\cap$ REC, then $A \in P^{T'}$ for some recursive tally set T'.*

Proof. Let $A \in \texttt{P/poly} \cap \texttt{REC}$ and $\mathcal{M}$ the Turing Machine that witnesses the fact that $A \in \texttt{REC}$. Furthermore, since $A \in \texttt{P/poly} = \bigcup_{T \in \texttt{tally}} P^T$, we conclude that $A \in P^T$ for some tally set T. We will now construct a recursive tally set T' such that $A \in P^{T'}$.

Let $\mathcal{M}'$ be the oracle Turing machine bounded in polynomial time n^k that decides A with oracle $T \subseteq 0^\star$. We assume that only 0's are written in the query tape and notice that the biggest word that might be queried to the oracle is $0^{|w|^k}$, where w is the input, since the running time is bounded by n^k and the machine can write at most one 0 per transition. Consider the function

$$f(n) = 1 + \sum_{i=1}^{n-1} (r^i + 1). \tag{12}$$

We construct a recursive tally set T' in the following way: for each $i \in \mathbb{N}$ we choose the least ζ_i such that $\mathcal{M}'$ working with oracle S_{ζ_i} agrees with $\mathcal{M}$ in every input with size i.[h] Such an index ζ_i must exist because we know that in the worst

[h]It is not needed to agree in inputs of different sizes since $\mathcal{M}''$ is built to consult a different S_{ζ_i} for each input size.

case scenario we can stop testing tally sets when S_{ζ_i} contains the (finite number of) words in T that are needed for answering queries when the input has size i. We then define

$$T'_i = \{0^{f(i)}y : y \in S_{\zeta_i}\} \text{ and } T' = \bigcup_{i=0}^{\infty} T'_i.$$

We want to show that T' is a recursive tally set such that $A \in \mathtt{P}^{T'}$. T' is clearly a tally set because for every $i \in \mathbb{N}$ we have that S_{ζ_i} is a tally set. To show that $A \in \mathtt{P}^{T'}$, we devise a machine $\mathcal{M}''$ according to Algorithm 4.

$\mathcal{M}''$ simulates the computation of $\mathcal{M}'$; however, whenever $\mathcal{M}'$ enters the query state, $\mathcal{M}''$ pads the query word with $f(|w|)$ 0's and performs its own call to T', proceeding as if $\mathcal{M}'$ would have the same answer. The only detail that could prevent $\mathcal{M}''$ from running in polynomial time was if the padding of the query words could not be done in a polynomial number of steps. This is not a problem since $f(n)$ is a polynomial of degree k which is fixed by $\mathcal{M}'$, therefore $\mathcal{M}''$ decides A in polynomial time with the help of oracle T'.

The function $f(n)$ used as a padding function grows fast enough to separate the words queried to T' for each input size: for any input w, the biggest word that $\mathcal{M}''$ can write in the query tape has size $f(|w|) + |w|^k$ which is smaller than $f(|w| + 1)$, which is the size of the smallest word than can be queried to the oracle with an

Algorithm 4

```
 1: procedure $\mathcal{M}''$
 2:     $w \leftarrow$ input
 3:     $n \leftarrow |w|$
 4:     $s' \leftarrow$ Start state of $\mathcal{M}'$ on input $w$          ▷ state of $\mathcal{M}'$
 5:     $s'' \leftarrow s'$                                                  ▷ state of $\mathcal{M}''$
 6: loop:
 7:     if $s'$ = halting state then return output of $\mathcal{M}'$
 8:     end if
 9:     if $s' \neq$ query state then
10:         $s' \leftarrow$ newt state of $\mathcal{M}'$                     ▷ continue the computation
11:         $s'' \leftarrow s'$                                              ▷ updating the state of $\mathcal{M}''$
12:     else
13:         $y \leftarrow$ word in query tape of $\mathcal{M}'$
14:         $p \leftarrow f(n)$                                              ▷ number of 0's to pad
15:         $w \leftarrow 0^p \cdot y$                                       ▷ padding the word
16:         $s'' \leftarrow$ result of asking $w$ to $T'$                    ▷ simulating the oracle on $\mathcal{M}''$
17:         $s' \leftarrow s''$                                              ▷ continue the computation as $\mathcal{M}'$
18:     end if
19: end loop
20: end procedure
```

input with size $|w| + 1$. In other words, if T'_{i_m} and T'_{i_M} are the smallest and the biggest word $\mathcal{M}''$ can write in the query tape for an input of size i, then we have

$$|T'_{1_m}| \leq |T'_{1_M}| < |T'_{2_m}| \leq |T'_{2_M}| < \cdots .$$

In particular, for every $i \neq j$ we have $T'_i \cap T'_j = \emptyset$. It is also important to notice that $\cup_{i=1}^{n} T'_i$ is a strict subset of $\cup_{i=1}^{n+1} T'_i$ since we can only add words that are larger than the ones we already have. We need the padding function to have this characteristic because there can be the case that the same tally word is asked by $\mathcal{M}'$ with inputs of different sizes. This raises a problem when building T' because we would have to append a word that could not be in T' or vice-versa.

Algorithm 5

```
1: **procedure** SEARCH OF $S_{\ell_i}$
2:     $i \leftarrow$ input
3:     $A \leftarrow \emptyset$
4:     $m \leftarrow 2^i$                                   ▷ $0^i$ is the $(2^i)^{th}$ word in lexicographic order
5:     **while** $m \leq 2^{i+1} - 1$ **do**                  ▷ $1^i$ is the $(2^{i+1} - 1)^{th}$ word in lexicographic order
6:         $w' \leftarrow m^{th}$ word in lexicographic order
7:         **if** $\mathcal{M}$ accepts $w'$ **then**
8:             $A \leftarrow A \cup \{w'\}$
9:         **end if**
10:        $m \leftarrow m + 1$
11:    **end while**
12:    $l \leftarrow 0$
13:    $bool \leftarrow$ false
14:    **while** $bool ==$ false **do**
15:        $j \leftarrow 2^i$
16:        $A' \leftarrow \emptyset$
17:        $l \leftarrow l + 1$
18:        **while** $j \leq 2^{i+1} - 1$ **do**
19:            $w'' \leftarrow j^{th}$ word in lexicographic order
20:            **if** $\mathcal{M}'_{S_l}$ accepts $w''$ **then**      ▷ $\mathcal{M}'_{S_l}$ is the machine $\mathcal{M}'$ working with oracle $S_l$
21:                $A' \leftarrow A' \cup \{w''\}$
22:            **end if**
23:        **end while**
24:        **if** $A == A'$ **then**
25:            $bool \leftarrow$ true
26:        **end if**
27:    **end while**
28:    **output** $S_l$
29: **end procedure**
```

It remains to be shown that T' is recursive. Given a tally word y', we can rewrite it as $y' = 0^{f(i)}y$ with the largest possible $i \in \mathbb{N}$; in this way, i and y are unique since we have $T'_i \cap T'_j = \emptyset$ for $i \neq j$. We search for the respective $S_{\zeta_{|w|}}$ using Algorithm 5 and by definition of T', we have that $y' \in T'$ if and only if $y \in S_{\zeta_{|w|}}$. □

Proposition 24. *$\mathtt{P/poly} \cap \mathtt{REC} \subseteq \mathtt{AP}(\mathtt{CI})$.*

Proof. Let $A \in \mathtt{P/poly} \cap \mathtt{REC}$. We can assume that A is decidable in polynomial time by a Turing machine using some tally set D as advice. Furthermore, by Proposition 23 we can assume that D is recursive. Consider again the time schedule

$$T(k) = \begin{cases} 2k+1, & \text{if } 0^k \in D \\ 2k, & \text{if } 0^k \notin D. \end{cases}$$

A similar reasoning to the one used in the proof of Proposition 19 shows that A can be decided in polynomial time by a SmSM using the physical oracle with unknown $y = 1/2$, infinite precision, and time schedule T. The only detail that differs is that D is recursive and therefore T is a total computable function. □

Proposition 25. *$\mathtt{AP}(\mathtt{CI}) = \mathtt{P/poly} \cap \mathtt{REC}$.*

Proof. It follows directly from Propositions 22 and 24. □

5. Conclusion

In previous papers, we have shown that machines that control experiments can decide in polynomial time the sets in non-uniform complexity classes that mix the central complexity classes P, PSPACE, and BPP with polynomial or logarithmic long advices. In [18] we uncover three types of experiments of measurement in Physics. Some values can be measured by successive approximations, approaching the unknown value by dyadic rationals above and below that value (see [14] for a universal measurement algorithm). Fundamental measurement of distance, angle, mass, etc., fall into this class. A second type of experiment was considered, e.g., the measurement of the threshold of a neuron in [7]. We can approach the desired value only from below the threshold. A third type of measurement was very recently discussed in [9].

The complexity classes characterised thus far are summarized in the Tables 1 and 2, for polynomial time and polynomial space bounds, respectively. Table 1 pictures the table of complexity classes of *polynomial time* Turing machines with different types of experiments as described above, considered with different concepts of precision and time tolerance. Two-sided experiments have been considered in [4, 5, 13, 14, 16, 17] and threshold experiments in [6, 7]. In the table "w/exponential time" stands for time of consultation exponential in the size of the query. Table 2 shows complexity classes of *polynomial space* Turing machines with different types of experiments (see [1]).

Table 1.

Type of Oracle		Infinite	Unbounded	Finite
Two-sided	lower bound	$\mathtt{P}/\log\star$	$\mathtt{BPP}/\log\star$	$\mathtt{BPP}/\log\star$
	upper bound	$\mathtt{P/poly}$	$\mathtt{P/poly}$	$\mathtt{P/poly}$
	upper bound (w/ exponential T)	—	—	—
Threshold	lower bound	$\mathtt{P}/\log\star$	$\mathtt{BPP}/\log\star$	$\mathtt{BPP}/\log\star$
	upper bound	—	—	—
	upper bound (w/ exponential T)	$\mathtt{P}/\log\star$	$\mathtt{BPP}/\log\star$	$\mathtt{BPP}/\log\star$
Vanishing Type 1 (Parallel)	lower bound	$\mathtt{P/poly}$	$\mathtt{P/poly}$	$\mathtt{BPP}/\log\star$
	upper bound	$\mathtt{P/poly}$	$\mathtt{P/poly}$	$\mathtt{BPP}/\log\star$
	upper bound (w/ exponential T)	—	—	—
Vanishing Type 2 (Clock)	lower bound	$\mathtt{P}/\log\star$	$\mathtt{BPP}/\log\star$	$\mathtt{BPP}/\log\star$
	upper bound	$\mathtt{P/poly}$	$\mathtt{P/poly}$	$\mathtt{BPP}/\log\star$
	upper bound (w/ exponential T)	—	$\mathtt{BPP}/\log\star$	—

Table 2.

	Infinite	Arbitrary	Fixed
Lower Bound with time schedule	PSPACE/poly	BPPSPACE/poly	BPPSPACE/poly
Lower Bound without time schedule	$2^{\{0,1\}^\star}$	$2^{\{0,1\}^\star}$	—
Upper Bound	$2^{\{0,1\}^\star}$	$2^{\{0,1\}^\star}$	BPPSPACE/poly

In this paper, we proved that the relevant classes P, P/poly, and P/poly $\cap$ REC can be characterised by the same SmSM machines considered in our previous investigation, controlling experiments in polynomial time as before, but now with protocols exhibiting non-computable time schedules. In other words, although the digital part of the SmSM's are standard Turing machines and the analog components are simulable on a Turing machine (since the vertices are placed at computable positions and the experimental time is computable), the non-computability arises from the non-controllable time interface between the digital and analog components of the SmSM machine. Although non-computability is expected in this case, is not straightforward that the computational power of the SmSM are the same as with computable protocols and non-simulable analog components.

Acknowledgements

The research of José Félix Costa is supported by FCT — Fundação para a Ciência e a Tecnologia, I.P., namely in the context of the I&D research unit

CFCUL — Centro de Filosofia das Ciências da Universidade de Lisboa, project FCT I.P.: UIDB/00678/2020 and UIDP/00678/2020. The research of Diogo Poças is supported by FCT via LASIGE Research Unit, ref. UIDB/00408/2020. We also thank to the anonymous referee.

References

[1] J. Alírio, J. F. Costa and L. Fonseca, Scatter machines bounded in space, in *Handbook of Unconventional Computing*, *Volume 1* (*Theory*), ed. A. Adamatzky, Chapter 3 (World Scientific, Singapore, 2021), pp. 59–97.

[2] T. Ambaram, E. Beggs, J. F. Costa, D. Poças and J. V. Tucker, An analogue-digital model of computation: Turing machines with physical oracles, in *Advances in Unconventional Computing*, *Volume 1* (*Theory*), ed. A. Adamatzky, *Emergence*, *Complexity and Computation*, Vol. 22 (Springer, 2016), pp. 73–115.

[3] J. L. Balcázar, J. Días and J. Gabarró, *Structural Complexity I*, 2nd edn. (Springer-Verlag, 1988, 1995).

[4] E. Beggs, J. F. Costa, B. Loff and J. V. Tucker, Computational complexity with experiments as oracles, *Proc. Royal Society*, *Series A* (*Mathematical*, *Physical and Engineering Sciences*) **464**(2098) (2008) 2777–2801.

[5] E. Beggs, J. F. Costa, B. Loff and J. V. Tucker, Computational complexity with experiments as oracles II. Upper bounds, *Proc. Royal Society*, *Series A* (*Mathematical*, *Physical and Engineering Sciences*) **465**(2105) (2009) 1453–1465.

[6] E. Beggs, J. F. Costa, D. Poças and J. V. Tucker, On the power of threshold measurements as oracles, in *Unconventional Computation and Natural Computation* (*UCNC 2013*), eds. G. Mauri, A. Dennunzio, L. Manzoni and A. E. Porreca, Lecture Notes in Computer Science, Vol. 7956 (Springer, 2013), pp. 6–18.

[7] E. Beggs, J. F. Costa, D. Poças and J. V. Tucker, Oracles that measure thresholds: The Turing machine and the broken balance, *J. Logic Comput.* **23**(6) (2013) 1155–1181.

[8] E. Beggs, J. F. Costa, D. Poças and J. V. Tucker, An analogue-digital Church–Turing thesis, *Int. J. Found. Comput. Sci.* **25**(4) (2014) 373–389.

[9] E. Beggs, J. F. Costa, D. Poças and J. V. Tucker, Computations with oracles that measure vanishing quantities, *Math. Struct. Comput. Sci.* **23**(6) (2017) 1155–1181.

[10] E. Beggs, P. Cortez, J. F. Costa and J. V. Tucker, A hierarchy for BPP//log* based on counting calls to an oracle, in *Emergent Computation* (*Festschrift for Selim Akl*), ed. A. Adamatzky, Emergence, Complexity and Computation, Vol. 21 (Springer, 2016), pp. 39–56.

[11] E. Beggs, P. Cortez, J. F. Costa and J. V. Tucker, Classifying the computational power of stochastic physical oracles, *Int. J. Unconvent. Comput.* **14**(1) (2018) 59–90.

[12] E. Beggs, J. F. Costa and J. V. Tucker, Computational models of measurement and Hempel's axiomatization, in *Causality*, *Meaningful Complexity and Knowledge Construction*, ed. A. Carsetti, Theory and Decision Library A, Vol. 46 (Springer, 2010), pp. 155–184.

[13] E. Beggs, J. F. Costa and J. V. Tucker, Physical oracles: The Turing machine and the Wheatstone bridge, *Studia Logica* **95**(1–2) (2010) 279–300.

[14] E. Beggs, J. F. Costa and J. V. Tucker, Limits to measurement in experiments governed by algorithms, *Math. Struct. Comput. Sci.* **20**(6) (2010) 1019–1050.

[15] E. Beggs, J. F. Costa and J. V. Tucker, The Turing machine and the uncertainty in the measurement process, in *Physics and Computation*, *P&C 2010*, ed. H. Guerra,

CMATI — Centre for Applied Mathematics and Information Technology (University of Azores, 2010), pp. 62–72.

[16] E. Beggs, J. F. Costa and J. V. Tucker, The impact of models of a physical oracle on computational power, *Math. Struct. Comput. Sci.* **22**(5) (2012) 853–879.

[17] E. Beggs, J. F. Costa and J. V. Tucker, Axiomatising physical experiments as oracles to algorithms, *Philosoph. Trans. Roy. Soc. Series A* (*Math. Phys. Engin. Sci.*) **370**(12) (2012) 3359–3384.

[18] E. Beggs, J. F. Costa and J. V. Tucker, Three forms of physical measurement and their computability, *Rev. Symbol. Logic* **7**(4) (2014) 618–646.

[19] G. A. Bekey and W. J. Karplus, *Hybrid Computation* (John Wiley & Sons, Inc., 1968).

[20] M. Davis, The myth of hypercomputation, in *Alan Turing*: *The Life and Legacy of a Great Thinker*, ed. C. Teuscher (Springer, 2006), pp. 195–212.

[21] M. Davis, Why there is no such discipline as hypercomputation, *Appl. Math. Comput.* **178**(1) (2006) 4–7.

[22] R. Carnap, *Philosophical Foundations of Physics* (Basic Books, 1966).

[23] R. Geroch and J. B. Hartle, Computability and physical theories, *Foundations of Physics* **16**(6) (1986) 533–550.

[24] C. G. Hempel, Fundamentals of concept formation in empirical science, *Int. Encyclopedia of Unified Science* **2**(7) (1952).

[25] H. T. Siegelmann, Computation beyond the Turing limit, *Science* (1995) 545–548.

[26] H. T. Siegelmann, *Neural Networks and Analog Computation*: *Beyond the Turing Limit* (Birkhäuser, 1999).

[27] H. T. Siegelmann and E. D. Sontag, Analog computation via neural networks, *Theoret. Comput. Sci.* **131**(2) (1994) 331–360.

[28] H. T. Siegelmann and E. D. Sontag, On the computational power of neural networks, *J. Comp. Syst. Sci.* **50**(1) (1995) 132–150.

[29] D. H. Krantz, P. Suppes, R. Duncan Luce and A. Tversky, *Foundations of Measurement* (Dover, 2009).

[30] J. von Neumann, Probabilistic logics and the synpaper of reliable organisms from unreliable components, in *Automata Studies* (Princeton University Press, 1956), pp. 43–98.

[31] A. Steven Younger, E. Redd, H. Siegelmann and C. Bell, A physical machine based on a super-Turing computational model (2017).