Journal of Computer and System Sciences 163 (2027) 103847

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

journal homepage: www.elsevier.com/locate/jcss

Piecewise rational affine maps, multidimensional generalized shifts, and computation over the real numbers

Daniel S. Graça [a,b,*]

[a] Universidade do Algarve, C. Gambelas, 8005-139 Faro, Portugal
[b] Center for Research and Development in Mathematics and Applications (CIDMA), Department of Mathematics, University of Aveiro, 3810-193 Aveiro, Portugal

ARTICLE INFO

Article history:
Received 9 December 2025
Received in revised form 23 June 2026
Accepted 14 August 2026
Available online 21 August 2026

Keywords:
Analog models of computation
Computation over the real numbers
Piecewise rational affine maps
Computable analysis
Church-Turing thesis

ABSTRACT

In this paper we study the computational power of certain classes of dynamical systems defined by the iteration of piecewise rational affine maps on the compact set $[0,1]^n$ as a model of computation over the real numbers. To achieve this aim we use symbolic dynamics to study the dynamics of this class of piecewise rational affine maps and in the process introduce multidimensional generalized shifts, which are an extension of the generalized shift of Moore to higher dimensional spaces. We show that multidimensional generalized shifts are not equivalent (conjugate) to (one-dimensional) generalized shifts. We then establish that computable analysis, which allows to extend computability based on Turing machines to real numbers, is computationally equivalent to multidimensional generalized shifts, both at a computability and at a computational complexity (polynomial time) level. This implies that a certain class of dynamical systems defined by the iteration of rational piecewise affine maps on the unit ball can thus compute any function computable over the real numbers in the computable analysis sense. These results highlight that several distinct models of computation over the real numbers, such as computable analysis, Claude Shannon's General Purpose Analog Computer, certain classes of piecewise rational affine maps, and the generalized shift, are all computationally equivalent.

© 2026 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Algorithms have been used since antiquity to solve several mathematical problems. Some examples are Euclid's algorithm to compute the greatest common divisor of two integers, the long division algorithm to compute the quotient and remainder of the Euclidean division of two integers, the simplex method for linear optimization, or several root-finding algorithms such as the bisection method or Newton's algorithm. In essence, an algorithm is a step-by-step procedure designed to obtain a certain result.

Given that so many problems can be solved with algorithms, a natural question is to ask which mathematical problems can be solved with algorithms.

Several researchers investigated this problem in the beginning of the 20th century and presented a variety of mathematical models intended to capture the intuitive notion of algorithm, such as recursive functions, Turing machine, lambda calculus, among others. It was soon proved that all these models were mathematically equivalent. These equivalence re-

* Correspondence to: Universidade do Algarve, C. Gambelas, 8005-139 Faro, Portugal.
E-mail address: dgraca@ualg.pt.

https://doi.org/10.1016/j.jcss.2026.103847
0022-0000/© 2026 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

sults supported the establishment of the so-called *Church-Turing thesis*, which states that the functions computable by an algorithm over discrete structures, such as the set of integers, are exactly those computable by Turing machines (or by any other equivalent mathematical model), thus providing a formalization for the notion of algorithm. Note that the Church-Thesis thesis cannot be proved, since there are no formal definitions of the term "algorithm". Instead, the Church-Turing thesis is intended to address this shortcoming by formalizing the notion of algorithm (effective method) over the integers.

Remarkably, once the notion of computable functions is made mathematically precise, e.g. using the formalism of Turing machines, it has been shown that several mathematical problems are *non-computable* in the sense that they are algorithmically unsolvable, i.e. their solutions cannot be obtained using algorithms. One such example of a non-computable problem is Hilbert's 10th problem [1], [2], that asks for an algorithm which can decide whether a polynomial Diophantine equation has a solution. Thus the Church-Turing thesis provides a unified framework for understanding whether a problem defined over a discrete structure can be solved with an algorithm.

Despite its success in the discrete domain, the Church-Turing thesis does not generalize to computation over continuous structures, such as the real numbers [3], [4], [5], [6]. This is a significant limitation given that many scientific and engineering problems are inherently grounded in continuous mathematics.

Initial steps towards addressing this gap have been taken through the development of models for real-number computation. Notable among these are computable analysis (see e.g. [7], [8], [9], [10]), which extends classical notions of computability to real numbers and real-valued functions, among other continuous structures, and traces its origin to Turing's seminal paper [11], and Shannon's General Purpose Analog Computer (GPAC) [12], [13], [14], a model inspired from earlier analog computers, usually known as Differential Analyzers [15]. It was shown that computable analysis and the GPAC are equivalent from a computability [16] and computational complexity [17] point of view, thus providing some first steps towards some variant of a Church-Turing thesis over the real numbers. Other results in that direction include the equivalence of some classes of real recursive functions [18] and some classes of computable functions in the computable analysis sense [19], [20], [21], [22], [23], [24], equivalence of the GPAC and some classes of real recursive functions [18], [14], equivalence of computable analysis with Feasible Real Random Access Machines [25], among others. Some related results can be found in e.g. [26], [27], [28], [29], [30], [3], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46]. The paper [6] provides a recent survey which includes many references pertinent to this topic.

However, it is not yet completely clear which models of computation over the real numbers are equivalent to computable analysis.

This paper aims to further the understanding of computation in continuous domains. In particular, we contribute to this line of inquiry by showing that a class of piecewise rational affine maps on the space $[0,1]^n$ is equivalent to computable analysis as a model of computation over the real numbers.

To achieve this aim we use symbolic dynamics. Symbolic dynamics is a standard tool of dynamical systems theory to study classes of maps such as Smale's horseshoe or the baker's map, by showing that these maps are conjugate to the shift operator over sequences of symbols (see e.g. [47]).

In particular, we extend the generalized shift introduced by Moore in [48], [49] to higher dimensions to study a particular class of piecewise rational affine maps. We show that this extension is proper in the sense that higher dimensional generalized shifts are not equivalent (conjugate) to the standard (one-dimensional) generalized shift.

Due to its similarity to the shift map, the generalized shift can also be used to analyze the behavior of dynamical systems defined by piecewise affine maps [49] or by hydrodynamics [50], [51], [52], [53], among others [54], via symbolic dynamics. In particular, the generalized shift can be used to show that certain systems exhibit a very complex behavior which is distinct from chaos, in the sense that it is non-computable [49].

Generalized shifts have not, to our knowledge, been previously used to compute with real numbers, although they have been considered as continuous models of computation for discrete entities (see e.g. [49], [50]). To achieve this aim, we show how to use the generalized shift as a model of computation over the real numbers and extend the generalized shift to more convenient (for our purposes) higher-dimensional variants. We show that higher-dimensional generalized shifts are equivalent to oracle Turing machines. We then use this result to show equivalence with computable analysis and present an equivalence result with a class of piecewise rational affine maps, which will imply that piecewise affine maps can compute any function over the reals which is computable in the computable analysis sense.

The paper is organized as follows. In Section 2 we review basic concepts about computation over discrete spaces such as Turing machines and the generalized shift. In Section 3 we review notions of computation over continuous spaces and, namely, computable analysis. In Section 4 we introduce a novel notion of multidimensional generalized shift and establish some connections with higher dimensional piecewise affine maps. In Section 5 we show that multidimensional generalized shifts are not equivalent (conjugate) to the classical (one-dimensional) generalized shift. In Section 6 we show that multidimensional shifts correspond to a class of piecewise affine maps, the Cantor piecewise affine maps. In Section 7 it is shown that oracle Turing machines are computationally equivalent to multidimensional generalized shifts. Section 8 introduces a notion of computation over the real numbers using multidimensional generalized shifts and Cantor piecewise affine maps, while Section 9 shows that computable analysis is equivalent to Cantor piecewise affine maps and to multidimensional generalized shifts over the real numbers.

2. Computation over discrete spaces

Before presenting the definition of Turing machine, we need some preliminary concepts.

An *alphabet* Σ is a finite set of elements called symbols. A word or string over Σ is a finite sequence $s_0, s_1, \ldots, s_k$ of symbols of Σ, which for simplicity we denote as $s_0 s_1 \ldots s_k$. By $|w|$, we denote the length of a word w, which is the number of symbols w has. There is a word ε with no symbols, the empty word. The set Σ^* is the set of all words over the alphabet Σ.

In some cases it may be convenient to order words from Σ^*. If we have an order $<$ defined over Σ, we can extend it to the *lexicographic order* $<_{\Sigma^*}$ over Σ^*, which we denote also as $<$ for simplicity, by using an idea akin to the alphabetical order used in dictionaries, with the sole difference that shorter words appear first. More formally, given two words $v, w \in \Sigma^*$, we have that:

1. If $|v| < |w|$ then $v < w$;
2. If $|v| = |w|$ and $v_i = w_i$ for all $i = 1 \ldots, k$ and $v_{k+1} < w_{k+1}$, then $v < w$.

2.1. Turing machines

The Turing machine is a mathematical model which can be used to formalize the intuitive notion of algorithm, and was introduced by Alan Turing [11]. Algorithms typically can be executed by a human using only paper and pencil/eraser. The paper is modeled by a one-dimensional tape divided into several cells. Since a human is only able to recognize finitely many symbols, due to aspects such as the finite resolution of the human retina [11, p. 249], and be in finitely many "states of mind" in its working memory (for similar reasons), it is assumed that a Turing machine works with finitely many distinct symbols which can be written consecutively on the tape, one symbol per cell, including a special symbol, the blank symbol B which just represents blank paper. The "states of mind" are represented by finitely many states of the Turing machine. Also, a human executing an algorithm can only direct its attention to a bounded portion of the tape, which in Turing machines is assumed to consist of only one cell. Then, depending on the current state and symbol being read on the cell under attention, it can decide to simultaneously update the state, update the symbol of the current cell under observation, and move its attention to the left or right of the cell under observation (or do not change the cell under observation). We can assume that the attention can be moved only one cell at a time. To indicate the cell under observation, in Turing machine one uses a tape head.

Formally:

Definition 2.1. A Turing machine M is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_f)$, where:

- Q is the set of states,
- $\Sigma \subseteq \Gamma$ is the input alphabet,
- Γ is the tape alphabet which contains a blank symbol $B \notin \Sigma$,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, 1\}$ is the transition function. $-1, 0, 1$ denote the following moves for the tape head: move one cell to the left, do not move, and move one cell to the right, respectively,
- q_0 is the initial state,
- q_f is the final state.

The Turing machine computes as follows. First the tape contains the input in consecutive cells, and all other cells contain the blank symbol, and it is assumed that the tape head is reading the leftmost symbol of the input. The computation starts on the initial state, and then the computation is done accordingly to the transition function, until eventually the final state is reached, indicating that the computation has finished. When that happens, the non-blank contents of the tape at the right of the tape head, including the symbol being read by the tape head, will be the output of the Turing machine. This allows us to associate to each Turing machine M a function $f_M : \Sigma^* \to \Sigma^*$, such that $f(w)$ gives the output of M on input w (if M never halts on input w, then $f(w)$ is undefined). We can encode elements of $\mathbb{N}$, $\mathbb{Z}$, etc., and other discrete structures as words over Σ by encoding natural numbers in binary/decimal format, etc. – see [55] for more details.

Given an input, the computation time of a Turing machine is the number of steps that the Turing machine uses until it halts. We also say that the function f is computable in time $O(g(n))$, for some $g : \mathbb{N} \to \mathbb{N}$, if there is some (fixed) $c \in \mathbb{N}$ such that $f(w)$ can be computed in time $\leq cg(|w|) + c$ for any input w.

At each step all the information about the current situation of the computation can be registered in a pair $(s, q) \in \Gamma^{\mathbb{Z}} \times Q$, where $\Gamma^{\mathbb{Z}}$ denotes the set of all infinite sequences $(s_i)_{i \in \mathbb{Z}}$ with $s_i \in \Gamma$. Here $s = (s_i)_{i \in \mathbb{Z}}$ is used to encode the tape contents, where s_0 denotes the symbol being read by the tape head, and q encodes the current state. The pair (s, q) is called a configuration.

We also remark that while we have defined a Turing machine with one tape, we could also define Turing machines with k tapes for some fixed $k \in \mathbb{N}$. These Turing machines work exactly like one-tape Turing machines, with the difference that the transition function must now be $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{-1, 0, 1\}^k$ to account for the k tapes and the initial input is

provided as before on the first tape and the output is provided in a specific tape, which can also be the first tape or another (fixed a priori) tape.

2.2. The generalized shift

The generalized shift was introduced by Moore in [48], [49] as a generalization of the shift map $\sigma : \Gamma^{\mathbb{Z}} \to \Gamma^{\mathbb{Z}}$ defined by $\sigma((s_i)_{i\in\mathbb{Z}}) = (s_{i+1})_{i\in\mathbb{Z}}$, which shifts all elements one position to the left. Note that the map σ^{-1} is well defined and is defined by $\sigma^{-1}((s_i)_{i\in\mathbb{Z}}) = (s_{i-1})_{i\in\mathbb{Z}}$. It shifts all elements of $(s_i)_{i\in\mathbb{Z}}$ one position to the right.

The generalized shift extends the classical shift by allowing that the amount and the direction of the shift depends on finitely many elements of $(s_i)_{i\in\mathbb{Z}}$. It also gives the possibility of modifying finitely many elements of $(s_i)_{i\in\mathbb{Z}}$. Formally:

Definition 2.2. Let A be an alphabet and let $s = (s_i)_{i\in\mathbb{Z}} \in A^{\mathbb{Z}}$. A generalized shift $\phi : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ is a map defined by $\phi(s) = \sigma^{F(s)}(G(s))$ where:

- **Shift:** $F : A^{\mathbb{Z}} \to \mathbb{Z}$ has a finite domain of dependence: there is a set $D_F = \{a, a+1, \ldots, a+b\} \subseteq \mathbb{Z}$ such that $F(s)$ only depends on the values of s_i for $i \in D_F$, i.e. $F(s) = F((s_i)_{i\in\mathbb{Z}}) = F((s_i)_{i\in D_F})$;
- **Substitution:** $G : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ has a finite domain of dependence and a finite domain of effect: if the finite domain of dependence is $D_G = \{c, c+1, \ldots, c+d\} \subseteq \mathbb{Z}$, then $G(s)$ only depends on the values of s_i for $i \in D_G$. Furthermore, if $D_e = \{m, m+1, \ldots, m+n\} \subseteq \mathbb{Z}$ is the finite domain of effect then the symbols of $G(s)$ are the symbols at the same position of s, except possibly for symbols with indices in D_e.

3. Computation over continuous spaces

Classical computation theory considers computability over discrete spaces such as Σ^* or $\mathbb{N}$. It is well known that there are explicit bijections between Σ^* and $\mathbb{N}$ (e.g. enumerate the words of Σ^* using the lexicographic order and correspond to each word its corresponding index in the ordered sequence of all strings of Σ^*) or even between $\mathbb{N}$ (and thus Σ^*) and other structures (see e.g. [56, pp. 26–27]) such as $\mathbb{N}^k$, $\mathbb{Z}^k$, or $\mathbb{Q}^k$ for any fixed $k \in \mathbb{N}$. Since there is a bijection between these sets and $\mathbb{N}$, all these sets are countable. In particular this allows us to use the standard theory of computation to compute over these sets.

However, it is known since Cantor that the cardinality of $\mathbb{R}$ is strictly greater than the cardinality of $\mathbb{N}$ and thus the set $\mathbb{R}$ is not countable. Therefore the classical approach has to be extended to be able to deal with real numbers. This is done in the next section.

3.1. Computable analysis

It is well known that $\mathbb{R}$ and $\mathbb{N}^{\mathbb{N}}$ have the same cardinality. The idea of computable analysis is to represent each real number as an element of $\mathbb{N}^{\mathbb{N}}$ and then to compute with real numbers by computing over their representations over $\mathbb{N}^{\mathbb{N}}$. This approach originates from Turing's seminal paper [11] where in the first paragraph he already considers computable real numbers. For example, one can consider a real number, for simplicity in $[0, 1]$, as a decimal number, where its decimal expansion is infinite (an integer can also be represented in that way, e.g. $1 = 1.000\ldots$). However, for topological reasons, decimal expansions are not well-suited to define computable functions over the real numbers. For example one can explore the identity $1.000\ldots = 0.999\ldots$ (two "faraway" sequences represent the same real number) to present a simple argument (see e.g. [10, Proposition 4.13]) that the function f satisfying $f(x) = 3x$ would not be computable under the decimal representation, by noting that an arbitrarily small difference between two inputs, such as the one between $0.333\ldots34\ldots$ and $0.333\ldots30\ldots$, could yield two very distinct outputs, seen as sequences of symbols, such as $1.00\ldots$ and $0.99\ldots$ in our previous example.

A better way to represent real numbers as elements of $\mathbb{N}^{\mathbb{N}}$ is, for example, to use the Cauchy sequence representation, where each real number is represented by a fast converging sequence of rational numbers. While there are several approaches to present computability theory over real numbers (see e.g. [7], [8], [10]), they are all equivalent. In this paper we use essentially the approach of [10].

Definition 3.1. A map $\psi : \mathbb{N} \to \mathbb{Q}$ is called a *name* for a real number $x \in \mathbb{R}$ if $|x - \psi(n)| \leq 2^{-n}$ for all $n \in \mathbb{N}$.

The above notion can be extended for points of $\mathbb{R}^k$ in a straightforward way. Namely, $\psi : \mathbb{N} \to \mathbb{Q}^k$ is a name for $x \in \mathbb{R}^k$ if $\|x - \psi(n)\| \leq 2^{-n}$ for all $n \in \mathbb{N}$.

Definition 3.2. A real number $x \in \mathbb{R}$ is computable if it has a computable name ψ.

In the above definition, the name $\psi : \mathbb{N} \to \mathbb{Q}$ is seen as computable in the classical (discrete) sense. The above definition can also be generalized in a straightforward way to computable real numbers of $\mathbb{R}^k$. One can also define the notion of computable open or closed subsets of $\mathbb{R}^n$, etc. (see e.g. [10]), but we will not explore this direction in this paper.

To define the notion of computable function $f : \mathbb{R}^n \to \mathbb{R}$, we need first to introduce oracle Turing machines. Here we use oracle Turing machines as they are especially well suited to consider complexity results [57]. In particular we use the approach of resource bounded oracle Turing machines introduced in [58] and used also e.g. in [59]. This approach allows one to define complexity respective to the "size" of the oracle encoding a real number, etc. Indeed, if we use an inefficient encoding of a real number, this will affect the time needed to get the result of a computation, even if the algorithm remains the same. So we want a measure of complexity which is invariant to the usage of more "inefficient" encodings (names).

Definition 3.3. An *oracle Turing machine* M is a multitape Turing machine that receives as input a word $w \in \Sigma^*$ and an *oracle* $\varphi : (\Gamma\backslash\{B\})^* \to (\Gamma\backslash\{B\})^*$. The Turing machine M with oracle φ is represented as M^φ. The oracle Turing machine also has two special tapes, called the *oracle query tape* and the *oracle answer tape*, and one special state, the *oracle state*. The oracle query tape can be queried by first writing the query, a string u, by next putting the tape head over the leftmost element of u, and then by entering the oracle state. Next it will replace the contents of the oracle answer tape by $\varphi(u)$, setting the tape head in this tape on the leftmost element of $\varphi(u)$. All this will take a single step. Other than this, the oracle Turing machine works as a standard Turing machine.

Oracle Turing machines allow the use of a function $\varphi : \mathbb{N} \to \mathbb{N}$ as an input. Note that considering a function $\varphi : (\Gamma\backslash\{B\})^* \to (\Gamma\backslash\{B\})^*$ or $\varphi : \mathbb{N} \to \mathbb{N}$ is computationally equivalent if we use a proper bijective encoding $\upsilon_{\Gamma\backslash\{B\}} : (\Gamma\backslash\{B\})^* \to \mathbb{N}$ such as the one based on the lexicographic order. Here we used two tapes for the oracle, one for the query, and another for the answer. This is a crucial addition provided by [58] when considering computational complexity. If one allows the same tape to be used for the query and for the answer, as usual in applications which do not involve computational complexity, we could iterate the calls to the oracle, which would not be suitable from a computational complexity perspective.

We also note that nothing prevents us from using oracle Turing machines with several oracles $\varphi_1, \ldots, \varphi_k$. In this case the oracle Turing machine will have k oracle query tapes, one for each oracle, which work exactly as the case with just one oracle, and k oracle answer tapes. We also note that the case of several oracles is computationally equivalent to the case of just one oracle, although perhaps with a cost at the computational complexity level. Namely, if 0 denotes a symbol not in Γ, then from $\varphi_1, \ldots, \varphi_k$ we can create a single oracle $\varphi : ((\Gamma \cup \{0\})\backslash\{B\})^* \to ((\Gamma \cup \{0\})\backslash\{B\})^*$ which simulates all the oracles $\varphi_1, \ldots, \varphi_k$ by taking

$$\varphi(0^i w) = \varphi_i(w) \text{ for all } i = 1, \ldots, k \text{ and } w \in \Gamma\backslash\{B\} \tag{1}$$

We can then simulate the oracle Turing machine with oracles $\varphi_1, \ldots, \varphi_k$ by an oracle Turing machine using the single oracle φ in a straightforward manner.

Definition 3.4. A function $f : \mathbb{R}^n \to \mathbb{R}$ is computable if there is an oracle Turing machine M such that given a name ψ of $x \in \mathbb{R}^n$ as an oracle, M^ψ computes a name of $f(x)$. In other words, M^ψ with input $n \in \mathbb{N}$ will return as output a rational number q satisfying $|f(x) - q| \leq 2^{-n}$ for all $n \in \mathbb{N}$.

Similarly, a function $f : \mathbb{R}^n \to \mathbb{R}^k$ is computable if all its components f_i, $i = 1, \ldots, k$, are computable, where $f(x) = (f_1(x), \ldots, f_k(x))$.

4. Multidimensional generalized shifts

The generalized shift over the alphabet A of Definition 2.2 is one-dimensional in the sense that its state space is $A^{\mathbb{Z}}$. In this section we generalize this notion to multidimensional generalized shifts, where the state space is $(A^{\mathbb{Z}})^k$.

Definition 4.1. Let A be a finite alphabet and let $k \in \mathbb{N}$. Consider an arbitrary element $s \in (A^{\mathbb{Z}})^k$, where $s = ((s_{i,1})_{i\in\mathbb{Z}}, \ldots, (s_{i,k})_{i\in\mathbb{Z}})$ and $s_{i,j} \in A$ for all $i \in \mathbb{Z}$ and $j = 1, \ldots, k$. Take also $\sigma^{(l_1,\ldots,l_k)}(s) = (\sigma^{l_1}((s_{i,1})_{i\in\mathbb{Z}}), \ldots, \sigma^{l_k}((s_{i,k})_{i\in\mathbb{Z}}))) \in (A^{\mathbb{Z}})^k$ for all $(l_1, \ldots, l_k) \in \mathbb{Z}^k$. A k-dimensional generalized shift $\phi : (A^{\mathbb{Z}})^k \to (A^{\mathbb{Z}})^k$ is a map defined by $\phi(s) = \sigma^{F(s)}(G(s))$ where:

- **Shift:** $F : (A^{\mathbb{Z}})^k \to \mathbb{Z}^k$ has a finite domain of dependence, i.e. there is a set $D_F = \{a, a+1, \ldots, a+b\}$ such that $F(s)$ only depends on the values of $s_{i,j}$ for $i \in D_F$ and $j \in \{1, \ldots, k\}$;
- **Substitution:** $G : (A^{\mathbb{Z}})^k \to (A^{\mathbb{Z}})^k$ has a finite domain of dependence and a finite domain of effect. That means that if the finite domain of dependence is $D_G = \{c, c+1, \ldots, c+d\}$, then $G(s)$ only depends on the values of $s_{i,j}$ for $i \in D_F$ and $j \in \{1, \ldots, k\}$. If $D_e = \{m, m+1, \ldots, m+n\}$ is the finite domain of effect this means that if $G(s) = \bar{s} = ((\bar{s}_{i,1})_{i\in\mathbb{Z}}, \ldots, (\bar{s}_{i,k})_{i\in\mathbb{Z}})$, then $\bar{s}_{i,j} = s_{i,j}$ whenever $i \notin D_e$, i.e. the only elements of s that can be changed by G are (eventually) the elements $s_{i,j}$ with $i \in D_e$ and $j \in \{1, \ldots, k\}$.

It should be clear from the definition that the (standard) generalized shift given by Definition 2.2 is simply a 1-dimensional generalized shift and vice versa.

Given some element $s \in (A^{\mathbb{Z}})^k$, where $s = ((s_{i,1})_{i\in\mathbb{Z}}, \dots, (s_{i,k})_{i\in\mathbb{Z}})$, the *components* of s are the elements $(s_{i,1})_{i\in\mathbb{Z}}, \dots, (s_{i,k})_{i\in\mathbb{Z}} \in A^{\mathbb{Z}}$. In Definition 4.1 we could have also assumed domains of dependence which depend on each component, instead of a domain of dependence which is the same for each component. However, this is equivalent to taking the maximum of the domains of dependence in each component and thus equivalent to Definition 4.1. This is because we can formally extend in each component the "smaller" domains of dependence, in the sense that the domain of dependence is extended by assuming that symbols not previously in the domain of dependence are now formally in this extended domain, until the maximum size of all domains of dependence is reached. Similarly, we can also assume that the domain of dependence is equal to the domain of effect for G (note again that G may map an element of $s_{i,j}$ in the domain of dependence to itself, without in practice changing it).

Remark 4.2. To simplify notation, if $s_i \in A$, we will often represent $s = (s_i)_{i\in\mathbb{Z}}$ as

$$s = \dots s_{-2}s_{-1}.s_0s_1s_2\dots \tag{2}$$

Moreover, without loss of generality, if a generalized shift has domain of dependence $\{-c, -c+1, \dots, d\}$ for $c, d > 0$, we can simplify the notation and write $F(s_{-c}s_{-c+1}\dots s_{-1}.s_0s_1\dots s_d)$ instead of

$$F(\dots s_{-c}s_{-c+1}\dots s_{-1}.s_0s_1\dots s_d\dots)$$

since the value of F only depends on $s_{-c}s_{-c+1}\dots s_{-1}.s_0s_1\dots s_d$. A similar approach can be used for G and for multidimensional generalized shifts.

Example 4.3. Let us consider $A = \{0, 1\}$ and consider the 2-dimensional generalized shift $\phi : \left(A^{\mathbb{Z}}\right)^2 \to \left(A^{\mathbb{Z}}\right)^2$ defined by $\phi(s) = \sigma^{F(s)}(G(s))$, where F and G have domain of dependence and of effect $\{0\}$ and

$$\begin{cases} F(.0,.0) = (0,1) \\ F(.0,.1) = (1,-1) \\ F(.1,.0) = (1,1) \\ F(.1,.1) = (1,1) \end{cases} \quad \text{and} \quad \begin{cases} G(.0,.0) = (.0,.0) \\ G(.0,.1) = (.0,.0) \\ G(.1,.0) = (.0,.1) \\ G(.1,.1) = (.1,.1) \end{cases}$$

Now let us pick $s = ((s_{i,1})_{i\in\mathbb{Z}}, (s_{i,2})_{i\in\mathbb{Z}})$, where $s_{i,1} = 1$ ($s_{i,2} = 0$) if i is odd and $s_{i,1} = 0$ ($s_{i,2} = 1$) if i is even, which for simplicity we denote as

$$s = (\dots 01.010101\dots, \dots 10.101010\dots)$$

or, in this particular case where there are repeating patterns, as

$$s = (\dots (01).01(01)\dots, \dots (10).10(10)\dots), \tag{3}$$

where the notation (01) near the ellipsis mean that the string 01 repeats itself in the direction of the ellipsis, i.e.

$$\dots (01).01(01)\dots = \dots 010101.010101010101\dots$$

and similarly for strings other than 01. Then we get

$$\phi(s) = (\dots (01)0.1(01)\dots, \dots (10)1.000(10)\dots).$$

5. Multidimensional generalized shifts are not conjugate to (one-dimensional) generalized shifts using a continuous encoding

In this section we explore whether a k-dimensional generalized shift ϕ, with $k > 1$, is conjugated to a (one-dimensional) generalized shift ψ in the sense that there is an injective encoding $enc : \left(A^{\mathbb{Z}}\right)^k \to A^{\mathbb{Z}}$ such that

$$enc \circ \phi = \psi \circ enc \tag{4}$$

We first note that the reverse direction is always true.

Lemma 5.1. *Let $k \in \mathbb{N}$. Then any generalized shift ϕ as defined in Definition 2.2 is conjugated to a k-dimensional generalized shift ψ in the sense (4).*

Proof. Let A be a finite alphabet and consider the infinite sequence $s = (s_i)_{i\in\mathbb{Z}}$, where $s_i \in A$. Let $\phi : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ be a generalized shift defined by $\phi(s) = \sigma^{F(s)}(G(s))$. Now define a k-dimensional shift $\psi : (A^{\mathbb{Z}})^k \to (A^{\mathbb{Z}})^k$ as $\psi(r) = \sigma^{\bar{F}(r)}(\bar{G}(r))$, where $r \in (A^{\mathbb{Z}})^k$ and the domains of dependence and the domain of effect of $\bar{F}$ and $\bar{G}$ are equal to the respective domains of dependence and domain of effect of F and G, respectively, and

$$\bar{F}(r) = (F((r_{i,1})_{i\in\mathbb{Z}}), F((r_{i,1})_{i\in\mathbb{Z}}), \dots, F((r_{i,1})_{i\in\mathbb{Z}}))$$
$$\bar{G}(r) = (G((r_{i,1})_{i\in\mathbb{Z}}), G((r_{i,1})_{i\in\mathbb{Z}}), \dots, G((r_{i,1})_{i\in\mathbb{Z}}))$$

so that ψ simulates ϕ on each of its k components. Now take $enc : A^{\mathbb{Z}} \to (A^{\mathbb{Z}})^k$ as

$$enc(s) = (s, \dots, s)$$

It is now trivial to check that (4) holds. □

We would now like to see if the converse of this lemma holds, i.e. whether any k-dimensional generalized shift is always conjugated to a generalized shift. This is not evident at first sight. It is well known that there are bijective maps from $\left(A^{\mathbb{Z}}\right)^2$ to $A^{\mathbb{Z}}$, or more generally from $\left(A^{\mathbb{Z}}\right)^k$ to $A^{\mathbb{Z}}$. For example, to obtain a bijective map from $\left(A^{\mathbb{Z}}\right)^2$ to $A^{\mathbb{Z}}$ one can associate

$$(\dots a_{-2}a_{-1}.a_0a_1a_2\dots, \dots b_{-2}b_{-1}.b_0b_1b_2\dots)$$

to

$$\dots a_{-2}b_{-2}a_{-1}b_{-1}.a_0b_0a_1b_1a_2b_2\dots \tag{5}$$

or more generally, we can obtain a bijective map $enc : \left(A^{\mathbb{Z}}\right)^k \to A^{\mathbb{Z}}$ defined by as

$$enc(((a_{i,1})_{i\in\mathbb{Z}}, \dots, (a_{i,k})_{i\in\mathbb{Z}})) = \dots a_{-1,1}a_{-1,2}\dots a_{-1,k}.a_{0,1}a_{0,2}\dots a_{0,k}a_{1,1}a_{1,2}\dots a_{1,k}\dots \tag{6}$$

However, these types of encoding do not work well for generalized shifts, in the case where each component shifts in opposite directions in a single step, as exemplified in the next lemma.

Theorem 5.2. *For any $k \geq 2$ there is a k-dimensional generalized shift ϕ for which there is no (one-dimensional) generalized shift ψ and no injective map $enc : \left(A^{\mathbb{Z}}\right)^k \to A^{\mathbb{Z}}$ such that (4) holds. More generally, there is no generalized shift ψ, no injective map $enc : \left(A^{\mathbb{Z}}\right)^k \to A^{\mathbb{Z}}$, and no $d, e \in \mathbb{N}$ such that*

$$enc \circ \phi^d = \psi^e \circ enc \tag{7}$$

Proof. We first note that we can suppose, without loss of generality (see comments after Definition 4.1) that the domain of dependence and effect of ϕ is $\{-m, \cdots, 0, \cdots, m\}$ for some $m \geq 0$ and that the amount of shift is also $\leq m$. Let $\{-k, \cdots, 0, \cdots, k\}$ be the domain of dependence/effect of a (one-dimensional) generalized shift ψ and suppose also without loss of generality that the amount of shift performed by ψ is always bounded by k. Then for any fixed $n \in \mathbb{N}$, ψ^n will also be a generalized shift with domain of dependence and effect $D_{m\cdot n} = \{-m \cdot n, \cdots, 0, \cdots, m \cdot n\}$ and shift bounded by $m \cdot n$. Thus, since the value of the generalized shift on the domain of dependence/effect only depends on the previous value on the domain of dependence/effect and the shift done by the generalized shift also only depends on the values with index in $D_{m\cdot n}$, we conclude that to ψ^n there is a finite set $B \subseteq \{(u, l, v) \in A^{D_{m\cdot n}} \times D_{m\cdot n} \times A^{D_{m\cdot n}}\}$ that to each word $u \in A^{D_{m\cdot n}}$ on the domain of dependence of ϕ, associates the new word v on the domain of effect by the application of ψ and the shift value $\leq m$.

Let $s \in A^{\mathbb{Z}}$ be a fixed point of ψ^n, i.e. s satisfies $\psi^n(s) = s$. Let $w = s_{-m\cdot n} \cdots s_{m\cdot n}$ and suppose that (w, j, v) is its associated element of B. Then $s = \psi^n(s)$ yields

$$s_i = \begin{cases} w_{i+j} & \text{if } i \in D_{m\cdot n} \\ s_{i+j} & \text{otherwise} \end{cases} \tag{8}$$

In other words, s must be j-periodic outside $D_{m\cdot n}$. But since $|j| \leq m \cdot n$ and because A is finite, there are only finitely many elements j-periodic sequences, and thus only finitely many elements s which satisfy (8). Therefore we conclude that for ψ^n can only have finitely many fixed points.

Now suppose without loss of generality that $k = 2$. A similar argument could be used if $k > 2$ by using dummy extra components. Consider the 2-dimensional generalized shift $\phi : \left(A^{\mathbb{Z}}\right)^2 \to \left(A^{\mathbb{Z}}\right)^2$ defined by

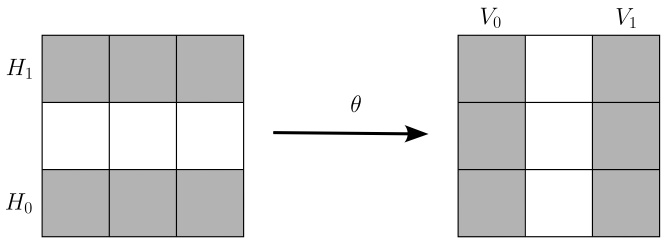

Fig. 1. The map θ applied to the unit square.

$$\phi(s,r) = (s, \sigma(r))$$

A similar argument as above shows that σ^n has at most $|A|^n$ fixed points. Furthermore $(s,r) \in \left(A^{\mathbb{Z}}\right)^2$ is a fixed point of ϕ^n if $\phi^n(s,r) = (s,r)$, which implies that

$$(s, \sigma^n(r)) = (s,r)$$

Since once we choose some r such that $\sigma^n(r) = r$, s can be arbitrary, we conclude that ϕ^n has infinitely many fixed points. But if (7) holds and (s,r) is a fixed point of ϕ^d, we get that

$$\begin{aligned} \psi^e \circ enc(s,r) &= enc \circ \phi^d(s,r) \\ &= enc(s,r) \end{aligned}$$

i.e. $enc(s,r)$ is a fixed point of ψ^e. Now, because enc is assumed to be injective, if $(s,r) \neq (\bar{s},\bar{r})$ are two distinct fixed points of ϕ^d, $enc(s,r) \neq enc(\bar{s},\bar{r})$ are also two fixed points of ψ^e. Hence, since ϕ^d has infinitely many fixed points, ψ^e also must have infinitely fixed points, which is a contradiction, since we have seen that ψ^s only has finitely many fixed points. □

6. Piecewise affine maps and multidimensional generalized shifts

In [49], Moore showed that any generalized shift is conjugated to a piecewise affine map on the Cantor set $\mathcal{C}^2 \subseteq \mathbb{R}^2$, and thus that any generalized shift can be simulated by a piecewise affine map (see [60] for a discussion on the notion of simulability). In this way, one can analyze the behavior of these piecewise affine maps on the Cantor set by using symbolic dynamics and the generalized shift, similarly to what is done to analyze the dynamics of Smale's horseshoe or the baker's map on the unit square via symbolic dynamics and the well known shift map. We now extend this result for k-dimensional generalized shifts, by extending the arguments of [49] to higher dimensions and defining exactly the class of piecewise affine maps that can be simulated by k-dimensional generalized shifts and vice versa.

Recall that the middle third Cantor set $\mathcal{C} \subseteq [0,1]$ can be constructed by deleting the open middle third of the unit interval and then proceeding recursively: let $C_0 = [0,1]$, and define recursively

$$C_{n+1} = \frac{C_n}{3} \cup \left(\frac{2}{3} + \frac{C_n}{3}\right)$$

for $n \geq 0$, where $C_n/3 = \{x/3 \in [0,1] : x \in C_n\}$ and $2/3 + C_n/3 = \{2/3 + x \in [0,1] : x \in C_n/3\}$. Then define

$$\mathcal{C} = \bigcap_{n=0}^{+\infty} C_n$$

Example 6.1. We now consider the baker's map $\theta : [0,1]^2 \to [0,1]^2$ given by

$$\theta(x,y) = \begin{cases} (x/3, 3y) & \text{if } y \leq 1/3 \\ \text{undefined} & \text{if } 1/3 < y < 2/3 \\ ((x+2)/3, 3y-2) & \text{if } 2/3 \leq y \end{cases}$$

(see Fig. 1). Let V_0 (V_1) be the left (right, respectively) vertical strip in square on the right of Fig. 1. Formally $V_0 = [0,1/3] \times [0,1]$ and $V_1 = [2/3,1] \times [0,1]$. Take $H_0 = \theta^{-1}(V_0)$ and $H_1 = \theta^{-1}(V_1)$, which will be the horizontal strips $[0,1] \times [0,1/3]$ and $[0,1] \times [2/3,1]$, denoted in the left square of Fig. 1. Note that

$$\theta^n([0,1]^2) = \theta^n(H_0 \cup H_1) = C_n \times [0,1] \text{ for } n \geq 1$$

is formed by 2^n vertical strips and

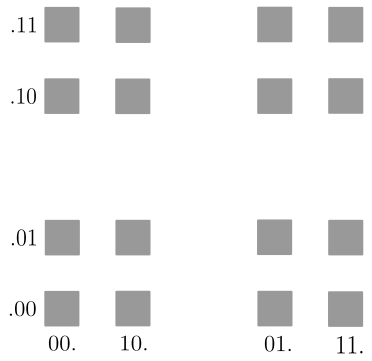

Fig. 2. Association of the connected components of C_2^2 to elements of $\{0,1\}^{\mathbb{Z}}$.

$$\theta^n(H_0 \cup H_1) = [0,1] \times C_{n+1} \text{ for } n \leq 0$$

is formed by 2^{n+1} horizontal strips. Therefore

$$\bigcap_{n=-\infty}^{+\infty} \theta^n([0,1]^2) = \mathcal{C} \times \mathcal{C} = \mathcal{C}^2.$$

Each element $X \in \mathcal{C}^2$ can be associated to its itinerary

$$S(X) = \ldots s_{-2}s_{-1}.s_0s_1s_2 \ldots \tag{9}$$

where

$$s_i = \begin{cases} 0 & \text{if } \theta^i(X) \in H_0 \\ 1 & \text{if } \theta^i(X) \in H_1 \end{cases}$$

for all $s_i \in \mathbb{Z}$ (see Fig. 1). Furthermore

$$S^{-1}((s_i)_{i\in\mathbb{Z}}) = \left(\sum_{i=1}^{\infty} \frac{2s_{-i}}{3^i}, \sum_{i=1}^{\infty} \frac{2s_{i-1}}{3^i} \right) \tag{10}$$

Note that if $X, Y \in \mathcal{C}^2$, then X and Y belong to the same small connected component of C_n^2 iff $(S(X))_i = (S(Y))_i$ for all $i = -n, \ldots, -1, 0, 1, \ldots, n-1$, as depicted in Fig. 2.

This implies that the map $\theta^i : C_n^2 \to [0,1]^2$, for all $i \in \{-n, \ldots, -1, 0, 1, \ldots, n-1\}$ is piecewise affine and that θ^i is topologically conjugated to the shift operator

$$S \circ \theta^i(X) = \sigma^i \circ S(X)$$

Note that, as remarked in [53], the shift operator can be implemented over a set A with $m > 2$ symbols using a piecewise affine map θ_m by using the Cantor set $\mathcal{C}_m$ instead of the middle third Cantor set, where the Cantor set $\mathcal{C}_m$ is obtained by taking

$$\mathcal{C}_m = \bigcap_{n=0}^{+\infty} C_{m,n} \tag{11}$$

where each interval of $C_{m,n+1}$ ($C_{m,0} = [0,1]$) is obtained by dividing each interval $C_{m,n}$ in $2m-1$ consecutive parts of equal length with indexes $1, 2, \ldots, 2m-1$, respectively, and by then removing the open parts with indexes $2, 4, \ldots, 2m$, to obtain $C_{m,n+1}$. By using a map $S_m : \mathcal{C}_m^2 \to A^{\mathbb{Z}}$ defined similarly to (9) we also get

$$S_m \circ \theta_m^i(X) = \sigma^i \circ S_m(X) \tag{12}$$

We first note that, similarly to what was done in [49], we can consider that each k-dimensional generalized shift ϕ over $\left(A^{\mathbb{Z}}\right)^k$ is conjugated in the sense (4) to another generalized shift ψ over $\left(\{0,1\}^{\mathbb{Z}}\right)^k$. To see this, suppose that A has n symbols and that $l \in \mathbb{N}$ is such that $2^l \geq n$. Then each element $s_i \in A$ can be encoded injectively into an element $enc(s_i) \in \{0,1\}^l$. Thus, given $s \in A^{\mathbb{Z}}$, we can encode it into $enc(s) \in \{0,1\}^{\mathbb{Z}}$ by encoding symbol by symbol into $\{0,1\}^l$. This

can be extended to dimension k, and it is a straightforward exercise to define the k-generalized shift ψ from ϕ which satisfies (4). So, on the remainder of this section, we suppose that $A = \{0, 1\}$.

Let us now assume, for simplicity and without loss of generality, that we are considering a k-generalized shift $\phi(s) = \sigma^{F(s)}(G(s))$ with domain of dependence equal to its domain of effect and which is assumed to be $\{-n, \dots, -1, 0, 1, \dots, n-1\}$. This can always be assumed by formally extending the domain of dependence and of effect of ϕ, i.e. by increasing n, until it includes both the original domain of dependence and effect. As seen above, we also assume without loss of generality that ϕ applies to binary sequences, i.e. elements of $\left(\{0,1\}^{\mathbb{Z}}\right)^k$.

Next we discuss how a piecewise affine map $\Psi : C_n^{2k} \to [0,1]^{2k}$ can be defined which is conjugated to the k-generalized shift ϕ. For simplicity, we call each connected component of C_n^{2k} a Cantor block. The intuition behind the map Ψ is similar to that used in [49]. Namely, since the domain of dependence and of effect of ϕ is $\{-n, \dots, -1, 0, 1, \dots, n-1\}$, its action will only depend on the elements $s_{-n,j} \dots s_{-1,j}.s_{0,j} \dots s_{n-1,j}$ $(j = 1, \dots, k)$ of $s = (s_i)_{i\in\mathbb{Z}} \in (\{0,1\}^{\mathbb{Z}})^k$, which corresponds to a Cantor block of C_n^{2k} via the encoding given by the baker's map. Then the action of ϕ in component j of s is done over a pair of coordinates in C_n^{2k}, namely the coordinates $2j-1$ and $2j$, in a way similar to what was done in Example 6.1. Since $\phi(s) = \sigma^{F(s)}(G(s))$, the action of G corresponds to a translation on C_n^{2k} and the action of $\sigma^{F(s)}$ corresponds to the application of the baker's map (possibly several times), similarly to Example 6.1.

More formally, let now see how G (see Definition 2.2) works componentwise. In component $j \in \{1, \dots, k\}$, it replaces the elements $s_{-n,j} \dots s_{-1,j}.s_{0,j} \dots s_{n-1,j}$ of the jth component of $s = (s_i)_{i\in\mathbb{Z}}$ by the elements $\bar{s}_{-n,j} \dots \bar{s}_{-1,j}.\bar{s}_{0,j} \dots \bar{s}_{n-1,j}$ provided by G (note that these latter values depend on all components of s). When this operation on the jth component of s is considered over the Cantor block, via the encoding provided by the baker's map, this corresponds to work with coordinates $2j-1$ and $2j$ over the Cantor block $X \subseteq C_n^{2k}$ associated to the address $s_{-n} \dots s_{-1}.s_0 \dots s_{n-1}$. The action of G corresponds to map the Cantor block X via a translation to the Cantor block $Y \subseteq C_n^{2k}$ associated to the address $\bar{s}_{-n} \dots \bar{s}_{-1}.\bar{s}_0 \dots \bar{s}_{n-1}$ of C_n^2. This translation, when restricted to coordinates $2j-1$ and $2j$ of X, is given by $\psi(x,y) = ((x - \alpha_{2j-1}) + \beta_{2j-1}, (y - \alpha_{2j}) + \beta_{2j})$ where

$$\begin{aligned} \alpha_{2j-1} &= \sum_{i=1}^{n} \frac{2s_{-i,j}}{3^i}, \quad \alpha_{2j} = \sum_{i=1}^{n} \frac{2s_{i-1,j}}{3^i}, \\ \beta_{2j-1} &= \sum_{i=1}^{n} \frac{2\bar{s}_{-i,j}}{3^i}, \quad \beta_{2j} = \sum_{i=1}^{n} \frac{2\bar{s}_{i-1,j}}{3^i} \end{aligned} \tag{13}$$

Now, assuming that $\left|F_j(s)\right| < n$ for all $j = 1, \dots, k$ (this can always be assumed by increasing n, if necessary), we note that $F(s)$ always takes the same value $r_{s_{-n}\dots s_0 \dots s_{n-1}} \in \mathbb{Z}^k$, which we denote as $r = (r_1, \dots, r_k)$ for simplicity. Then $\sigma^{F_j(s)} = \sigma^{r_j}$ for the jth component of the k-dimensional generalized shift ϕ. The action of $\sigma^{F_j(s)} = \sigma^{r_j}$ can thus be implemented over the coordinates $2j-1$ and $2j$ of $Y \subseteq C_n^{2k}$ by applying θ^{r_j}, similarly to what was done in Example 6.1. Thus, the jth component of the generalized shift ϕ can be implemented on coordinates $2j-1$ and $2j$ of the Cantor block $X \subseteq C_n^{2k}$ with address $s_{-n} \dots s_{-1}.s_0 \dots s_{n-1}$ by taking the affine map

$$((\Psi(X))_{2j-1}(x), (\Psi(X))_{2j}(x)) = \theta^{r_j} \circ \psi$$

Therefore if $r_j > 0$ ($r_j < 0$) $\theta^{r_j} \circ \psi$ corresponds to applying a linear transformation that stretches (compresses, respectively) the $2j$ coordinate of $X \subseteq C_n^{2k}$ by a factor of 3^{r_j} and compresses (stretches, respectively) the $2j-1$ coordinate of X by a factor of 3^{r_j}, and may add constants over the coordinates of $2j-1$ and $2j$ of the Cantor block $X \subseteq C_n^{2k}$ (the constants may differ on coordinates $2j-1$ and $2j$ of $X \subseteq C_n^{2k}$). If $r_j \geq 0$, this yields the following for the coordinate $2j-1$ of X (note that the term $\sum_{i=1}^{r_j} \frac{2\bar{s}_{i-1,j}}{3^{r_j-i+1}}$ comes from the coordinate $2j$ due to the shift applied to the jth component of s)

$$\begin{aligned} (\Psi(X))_{2j-1}(x) &= 3^{-r_j}\left((x_{2j-1} - \alpha_{2j-1}) + \beta_{2j-1}\right) + \sum_{i=1}^{r_j} \frac{2\bar{s}_{i-1,j}}{3^{r_j-i+1}} \\ &= 3^{-r_j}\left(x_{2j-1} - \sum_{i=1}^{n} \frac{2s_{-i,j}}{3^i} + \sum_{i=1}^{n} \frac{2\bar{s}_{-i,j}}{3^i}\right) + \sum_{i=1}^{r_j} \frac{2\bar{s}_{i-1,j}}{3^{r_j-i+1}} \\ &= 3^{-r_j}\left(x_{2j-1} - \sum_{i=1}^{n} \frac{2s_{-i,j}}{3^i}\right) + \left(\sum_{i=1}^{n} \frac{2\bar{s}_{-i,j}}{3^{i+r_j}} + \sum_{i=1}^{r_j} \frac{2\bar{s}_{i-1,j}}{3^{r_j-i+1}}\right) \\ &= 3^{-r_j}\left(x_{2j-1} - \sum_{i=1}^{n} \frac{2s_{-i,j}}{3^i}\right) + \sum_{i=1}^{n+r_j} \frac{2\bar{s}_{r_j-i,j}}{3^i} \end{aligned}$$

and for coordinate $2j$ we get

$$(\Psi(X))_{2j}(x) = 3^{r_j}\left(x_{2j} - \sum_{i=1}^{n}\frac{2s_{i-1,j}}{3^i} + \sum_{i=1}^{n}\frac{2\bar{s}_{i-1,j}}{3^i}\right) - \sum_{i=1}^{r_j}\frac{2\bar{s}_{i-1,j}}{3^{i-r_j}}$$
$$= 3^{r_j}\left(x_{2j} - \sum_{i=1}^{n}\frac{2s_{i-1,j}}{3^i}\right) + \sum_{i=r_j+1}^{n}\frac{2\bar{s}_{i-1,j}}{3^{i-r_j}}$$

Proceeding similarly for all other Cantor blocks of C_n^{2k}, we obtain a map $\Psi : C_n^{2k} \to [0,1]^{2k}$ which is piecewise affine and satisfies

$$S \circ \Psi^i(X) = \phi^i \circ S(X) \tag{14}$$

We also note that in this argument it makes no difference, when considering one component of a k-dimensional generalized shift, if both $F(s)$ and $G(s)$ (considered only over this component) depend not only on this component, as a one-dimensional generalized shift but also on all the remaining components.

Moreover, from the above arguments, the class of piecewise affine maps which satisfies (14) is exactly the following.

Definition 6.2. A Cantor piecewise affine map is a map $\Psi : C_n^{2k} \to [0,1]^{2k}$ which is affine on each Cantor block (i.e. connected component of C_n^{2k}). Furthermore the image $\Psi(X)$ of a Cantor block X of C_n^{2k} is required to have the property that each pair $(\Psi(X))_{2j-1}, (\Psi(X))_{2j}$, where $j = 1, \ldots, k$, is associated to a value $r_j \in \{-n, \ldots, -1, 0, 1, \ldots, n-1\}$ such that for every $x = (x_1, \ldots, x_{2n}) \in X$ one has

$$(\Psi(X))_{2j-1}(x) = \begin{cases} (x_{2j-1} - \alpha_{2j-1})\,3^{-r_j} + \bar{\beta}_{2j-1} & \text{if } r_j \geq 0 \\ (x_{2j-1} - \alpha_{2j-1})\,3^{|r_j|} + \underline{\beta}_{2j-1} & \text{if } r_j < 0 \end{cases}$$
$$(\Psi(X))_{2j}(x) = \begin{cases} (x_{2j} - \alpha_{2j})\,3^{r_j} + \bar{\beta}_{2j} & \text{if } r_j \geq 0 \\ (x_{2j} - \alpha_{2j})\,3^{-|r_j|} + \underline{\beta}_{2j} & \text{if } r_j < 0 \end{cases}$$

where α_{2j-1} and α_{2j} are given by (13) and

$$\bar{\beta}_{2j-1} = \sum_{i=1}^{n+r_j}\frac{2\bar{s}_{r_j-i,j}}{3^i}, \quad \underline{\beta}_{2j-1} = \sum_{i=|r_j|+1}^{n} 2\bar{s}_{-i,j}3^{|r_j|-i}$$
$$\bar{\beta}_{2j} = \sum_{i=r_j+1}^{n} 2\bar{s}_{i-1,j}3^{r_j-i}, \quad \underline{\beta}_{2j} = \sum_{i=1}^{n+|r_j|}\frac{2\bar{s}_{i-|r_j|-1,j}}{3^i}$$

with $\bar{s}_{i,j} \in \{0,1\}$ for all $i \in \{-n, \ldots, -1, 0, 1, \ldots, n-1\}$ and all $j = 1, \ldots, k$.

From the above we conclude the following result.

Theorem 6.3. *Any k-dimensional generalized shift ϕ defined over the alphabet $\{0,1\}$ is conjugated to a Cantor piecewise affine map $\Psi : C_n^{2k} \to [0,1]^2$ on the Cantor set $C^{2k} \subseteq \mathbb{R}^{2k}$, for some $n \in \mathbb{N}$, via the map S given by (14), i.e.*

$$S \circ \Psi = \phi \circ S \tag{15}$$

Reciprocally, any Cantor piecewise affine map $\Psi : C_n^{2k} \to [0,1]^{2k}$ is conjugated to a k-dimensional generalized shift ϕ defined over the alphabet $\{0,1\}$ in the sense (15).

We note that the above definition and result can be extended to alphabets A with $m = |A| > 2$ symbols by considering the Cantor set $\mathcal{C}_m$ given by (11). We can assume, without loss of generality, that the m elements of A correspond injectively to the elements of $\{0, 1, \ldots, m-1\}$. For simplicity, below we identify directly the elements of A to their counterpart in $\{0, 1, \ldots, m-1\}$.

Definition 6.4. An m-Cantor piecewise affine map, where $m \geq 2$ is a map $\Psi : C_{m,n}^{2k} \to [0,1]^{2k}$ which is affine on each Cantor block of $C_{m,n}^{2k}$. Furthermore, the image $\Psi(X)$ of a Cantor block X of $C_{m,n}^{2k}$ is required to have the property that each pair $(\Psi(X))_{2j-1}, (\Psi(X))_{2j}$, where $j = 1, \ldots, k$, is associated to a value $r_j \in \{-n, \ldots, -1, 0, 1, \ldots, n-1\}$ such that for every $x = (x_1, \ldots, x_{2n}) \in X$ one has

$$(\Psi(X))_{2j-1}(x) = \begin{cases} (x_{2j-1} - \alpha_{2j-1})(2k-1)^{-r_j} + \bar{\beta}_{2j-1} & \text{if } r_j \geq 0 \\ (x_{2j-1} - \alpha_{2j-1})(2k-1)^{|r_j|} + \underline{\beta}_{2j-1} & \text{if } r_j < 0 \end{cases}$$

$$(\Psi(X))_{2j}(x) = \begin{cases} (x_{2j} - \alpha_{2j})(2k-1)^{r_j} + \bar{\beta}_{2j} & \text{if } r_j \geq 0 \\ (x_{2j} - \alpha_{2j})(2k-1)^{-|r_j|} + \underline{\beta}_{2j} & \text{if } r_j < 0 \end{cases}$$

where

$$\alpha_{2j-1} = \sum_{i=1}^{n} \frac{2s_{-i,j}}{(2m-1)^i}, \quad \alpha_{2j} = \sum_{i=1}^{n} \frac{2s_{i-1,j}}{(2m-1)^i},$$

and

$$\bar{\beta}_{2j-1} = \sum_{i=1}^{n+r_j} \frac{2\bar{s}_{r_j-i,j}}{(2m-1)^i}, \quad \underline{\beta}_{2j-1} = \sum_{i=|r_j|+1}^{n} 2\bar{s}_{-i,j}(2m-1)^{|r_j|-i}$$

$$\bar{\beta}_{2j} = \sum_{i=r_j+1}^{n} 2\bar{s}_{i-1,j}(2m-1)^{r_j-i}, \quad \underline{\beta}_{2j} = \sum_{i=1}^{n+|r_j|} \frac{2\bar{s}_{i-|r_j|-1,j}}{(2m-1)^i}$$

with $\bar{s}_{i,j} \in \{0, 1, \ldots, k-1\}$ for all $i \in \{-n, \ldots, -1, 0, 1, \ldots, n-1\}$ and all $j = 1, \ldots, k$.

From the above we conclude the following result.

Theorem 6.5. *Any k-dimensional generalized shift ϕ defined over the alphabet A with m symbols is conjugated to a m-Cantor piecewise affine map $\Psi : C_{m,n}^{2k} \to [0,1]^{2k}$ on the Cantor set $C_m^{2k} \subseteq [0,1]^{2k}$, for some $n \in \mathbb{N}$, via the map $S_m : C_m^2 \to A^{\mathbb{Z}}$ which appears in (12), i.e.*

$$S_m \circ \Psi = \phi \circ S_m \tag{16}$$

Reciprocally, any m-Cantor piecewise affine map $\Psi : C_n^{2k} \to [0,1]^{2k}$ is conjugated to a k-dimensional generalized shift ϕ defined over an alphabet A with m elements in the sense (16).

7. Equivalence between multidimensional generalized shifts and oracle Turing machines

In this section we show that the generalized shift is computationally equivalent to oracle Turing machines, both at the computability and computational complexity level.

We first start by showing that an oracle Turing machine can be simulated by a generalized shift. To achieve this aim, we first need a preliminary definition. Later on, in this section, we will show the converse direction, i.e. that generalized shifts can be simulated by oracle Turing machines.

We notice that, when considering complexity over oracle Turing machines, the time needed to compute may depend on the size of the answer provided by some oracle query, e.g. when we need to read the whole answer of the oracle to decide the next action to do. Hence, the complexity may be dependent on the "size" of the oracle. This is explored, for example, in [58], [57], [59]. We can define the size of an oracle as follows.

Definition 7.1. Let $\varphi : \Sigma_1^* \to \Sigma_2^*$ where Σ_1, Σ_2 are alphabets. We say that φ is bounded by f or that f is a bound for φ, where $f : \mathbb{N} \to \mathbb{N}$ is a non-decreasing function, if $|\varphi(w)| \leq f(|w|)$ for all $w \in \Sigma^*$. We also define $|\varphi| : \mathbb{N}_0 \to \mathbb{N}_0$ as $|\varphi|(n) = \max_{|w| \leq n} |\varphi(w)|$.

Theorem 7.2. *Let M be an oracle Turing machine with k tapes, which uses one oracle of the type $\varphi : \Sigma_1^* \to \Sigma_2^*$, where Σ_1 is a unary alphabet (i.e. consisting of one symbol). Then M can be simulated by a $(k+2)$-dimensional generalized shift in time $O(n^2 \cdot f(n))$, where f is a bound for the oracle provided to M and n is the number of steps of M to be simulated.*

Proof. Let M be an oracle Turing machine with k tapes and tape alphabet Σ, where $B \in \Sigma$ is the blank symbol, and which uses an oracle $\varphi : \Sigma_1^* \to \Sigma_2^*$. This oracle Turing machine uses at least three tapes, one for the oracle query, another for the oracle answer, and another which initially contains the input $x \in \Gamma^*$, where $\Gamma \subseteq \Sigma$ is the input alphabet. We now construct a $(k+2)$-dimensional generalized shift ϕ which receives its input $w \in \Gamma^*$ on its first component and the oracle on its $k+1$ component, in a way which is somewhat inspired on the analog shift map presented in [26] and on the results from [53]. Namely, given some arbitrary $w \in \Gamma^*$, $\varphi : \Sigma_1^* \to \Sigma_2^*$, let $s[w, \varphi] \in (\Sigma^{\mathbb{Z}})^{k+2}$ be such that $s_1[w, \varphi] = \ldots(B)B.wB(B)\ldots$, $s_2[w, \varphi] = \ldots = s_k[w, \varphi] = \ldots(B)B.B(B)\ldots$, and

$$s_{k+1}[w,\varphi] = \ldots (B)\blacklozenge.\varphi(\varepsilon)B\varphi(0)B\varphi(00)B\varphi(000)\ldots \tag{17}$$

where, without loss of generality, the only symbol in Σ_1 is the symbol 0, i.e. we take $\Sigma_1 = \{0\}$, and $\blacklozenge \notin \Sigma_2$ marks the beginning of the encoding of the oracle. The last component $s_{k+2}[w,\varphi]$ of ϕ will be used as a state, which is encoded as a string of fixed length. The component 1 of ϕ will simulate the first tape of the oracle Turing machine M, which is assumed to have the input. Namely, the symbol being read by the tape head of M in this tape is a_0 if the first component is $\ldots (B)Ba_{-m}\ldots a_{-1}.a_0\ldots a_l B(B)\ldots$. Similarly, the other components $s_2[w,\varphi] = \ldots = s_k[w,\varphi]$ will simulate the remaining $k-1$ tapes of M. For simplicity, we will sometimes refer to those components also as tapes. Let us also assume, without loss of generality, that the tape $k-1$ corresponds to the oracle query tape and that the tape k corresponds to the oracle answer tape. Then based on the finite amount of data given by: (i) the symbol at the right of the dot on each tape (i.e. the symbol being read by the tape head in the respective tape of M) and (ii) the state written on component $k+2$, we can define the generalized shift ϕ so that it updates the state in tape $k+2$ according to the transition rule of M and updates the first $k-1$ components (i.e. tapes of M) to mimic the changes in the tapes of M provided by the transition function of M by updating the symbol with index 0 in each of the first components and by shifting an appropriate amount (-1, 0, or 1, depending on the case) to mimic the tape head move (instead of moving the tape head, we can assume it to be fixed and instead shift the tape). In this way we are able to simulate an ordinary k-tape Turing machine with a $(k+1)$-dimensional generalized shift in real time (just dimension $k+1$ and not $k+2$, since we are not using the oracle, and thus do not need the former component $k+1$ given by (17)). What remains to be done is to consider the oracle.

In the oracle Turing machine M, the Turing machine can enter the oracle state with input w (which is in tape $k-1$) and replaces any non-blank string of tape k by $\varphi(w)$ in just one step. Suppose that the oracle Turing machine (generalized shift) just entered the oracle state. To mimic the oracle query by the generalized shift, we use component $k-1$ of the generalized shift to simulate the query to the oracle done in the oracle tape of the Turing machine. The content of this tape has the format 0^n just before calling the oracle, and then the generalized shift performs several operations, as described below, to read $\varphi(0^n)$ in component $k+1$ of the generalized shift (which encodes the oracle) and then replaces the previous non-blank string by $\varphi(0^n)$ in component k. In other words, the simulation of Turing machines (without oracles) can be done in real time with a generalized shift, but the oracle query involves a computational overhead when simulated by the generalized shift. More concretely, the generalized shift ϕ simulates an oracle query as follows. First, it deletes the previous content of tape k. Note that before deletion tape k is either empty, if no previous oracle call was made, or it has the string $\varphi(0^j)$ for some $j < n$ if up to now n steps of the simulation have been performed. Since $|\varphi(0^j)| \leq f(j) \leq f(n)$, deleting the previous string $\varphi(0^j)$ can be done in time $O(f(n))$.

After deleting the previous content of tape k, we want to update its content to $\varphi(0^n)$. This is done as follows. If $n = 0$, the contents of component $k-1$ is

$$\ldots (B)B.B(B)\ldots \tag{18}$$

the generalized shift ϕ copies $\varphi(\varepsilon)$ from the component $k+1$ to the component k to get

$$\ldots (B)B.\varphi(\varepsilon)(B)\ldots$$

Similarly, if the contents of component $k-1$ is

$$\ldots (B)B.\underbrace{0\ldots 0}_{n \text{ times}}(B)\ldots \tag{19}$$

then ϕ passes (by shifting this component left) over $\varphi(\varepsilon)$ in component $k+1$ and once it reaches a B in component $k+1$ it shifts component $k-1$ one position to the left, yielding

$$\ldots (B)B\underbrace{0.0\ldots 0}_{n \text{ times}}(B)\ldots$$

and then starts reading $\varphi(0)$ in component $k+1$ until it reaches a B and then similarly it again shifts component 2 one position to the left yielding

$$\ldots (B)B\underbrace{00.0\ldots 0}_{n \text{ times}}(B)\ldots$$

on component 2. This procedure continues until we get

$$\ldots (B)B\underbrace{000\ldots 0}_{n \text{ times}}.(B)\ldots$$

on component 2. At this step we will have

$$\ldots (B)B\blacklozenge\varphi(\varepsilon)B\varphi(0)B\varphi(00)\ldots B.\varphi(0^n)B\ldots$$

on component $k+1$ and we can copy $\varphi(0^n)$ to replace the contents of component k, which will thus have the result $\varphi(0^n)$ of the output. Next we shift the component $k+1$ right until we get (17) on component $k+1$, by ensuring that the marker ♦ of the beginning of the encoding of the oracle is in position -1 and thus "reset" this component. At the same time we also shift the component $k-1$ until we get (19) for this component. The number of steps (iterations) that the generalized shift ϕ needs to simulate the oracle query is thus $O(n \cdot f(n))$, which is the time needed to read $\varphi(\varepsilon), \varphi(0), \ldots, \varphi(0^n)$ plus the blank symbols B and to return back. Therefore, the time the generalized shift ϕ needs to simulate n steps of M is $O(n^2 \cdot f(n))$. □

We note that in the previous theorem we can reduce the dimension of the generalized shift from $k+2$ to $k+1$.

Corollary 7.3. *Let M be an oracle Turing machine with k tapes, which uses one oracle of the type $\varphi : \Sigma_1^* \to \Sigma_2^*$, where Σ_1 is a unary alphabet. Then M can be simulated by a $(k+1)$-dimensional generalized shift in time $O(n^2 \cdot f(n))$, where f is a bound for the oracle provided to M.*

Proof. To prove this corollary we proceed as in the proof of Theorem 7.2. However, instead of using the component $k+2$ to store a state q seen as a string of bounded length $\leq M$ for some $M > 0$, we replace the contents (2) of component 1 by

$$\ldots a_{-2}a_{-1}.qa_0a_1a_2\ldots$$

just as done by Moore in [49]. By extending the alphabet to include states and the domain of dependence and effect of ϕ, and by making straightforward changes to maintain the current state after the dot, we no longer need the component $k+2$ to keep track of the computation and thus this component can be eliminated. □

Corollary 7.4. *Let M be an Turing machine with k tapes. Then M can be simulated by a k-dimensional generalized shift in real time.*

Proof. Proceed as in the proof of Corollary 7.3, but without using the oracle. □

Corollary 7.5. *Let M be an oracle Turing machine with k tapes, which uses one oracle of the type $\varphi : \Sigma_1^* \to \Sigma_2^*$, where the alphabet Σ_1 has $A > 1$ elements. Then M can be simulated by a $(k+1)$-dimensional generalized shift in time $O(n \cdot A^n \cdot f(n))$, where f is a bound for the oracle provided to M.*

Proof. Proceed as in the proof of Corollary 7.3, but now notice that the oracle will be coded in component $k+1$ according to the lexicographic order on the argument of φ, i.e. if w_2 is the successor of w_1 according to the lexicographic order, then the component $k+1$ will be

$$\ldots(B)♦.\varphi(\varepsilon)B\ldots B\varphi(w_1)B\varphi(w_2)B\ldots$$

Thus when querying a word w of size n, we need to check $\leq A^{n+1}$ elements until finding $\varphi(w)$. □

Corollary 7.6. *Let M be an oracle Turing machine with k tapes, which uses $m \geq 1$ oracles of the type $\varphi : \Sigma_1^* \to \Sigma_2^*$, where the alphabet Σ_1 has $A > 1$ elements. Then M can be simulated by a $(k+1)$-dimensional generalized shift in time $O(n \cdot A^n \cdot f(n))$, where f is a bound for the oracles provided to M.*

Proof. Proceed as in the proof of Corollary 7.5, but use just one oracle φ which encodes all the remaining m oracles as done in (1). The only difference is that in the component encoding the oracle φ we do not order the words $\varphi(0^i w)$ according to the lexicographic order on its argument, but as follows: if w_2 is the successor of w_1 according to the lexicographic order, then the component $k+1$ will be

$$\ldots(B)♦.\varphi(0\varepsilon)B\varphi(00\varepsilon)\ldots\varphi(0^m\varepsilon)B\ldots B\varphi(0w_1)B\ldots B\varphi(0^m w_1)B\varphi(0w_2)B\ldots.$$

A straightforward analysis of the time needed to simulate the oracle Turing machine shows that this is done in time $O(n \cdot A^n \cdot f(n))$ by the generalized shift. □

Now that we have shown that oracle Turing machines can be simulated with generalized shifts, we prove the converse direction. We first notice that, given some initial state $s \in (A^{\mathbb{Z}})^k$, at each iteration of a generalized shift ϕ, only finitely many symbols of s are changed on each component, and that the amount of shift done on each component is also bounded on each iteration. Thus, we say that an oracle Turing machine simulates ϕ, if given an oracle which encodes s and some input $n \in \mathbb{N}$, it outputs the positions of the symbols changed on each component of $\phi^n(s)$ relative to s, as well as the changed symbols and the total shift performed on each component.

Theorem 7.7. *Let ϕ be a k-dimensional generalized shift ϕ. Then ϕ can be simulated in time $O(n)$ by an oracle Turing machine with $3k$ tapes.*

Proof. We create an oracle Turing machine M with k oracles and $3k$ tapes that behaves as follows. Let $s \in (A^{\mathbb{Z}})^k$ be the initial state (input) of the k-dimensional generalized shift ϕ and assume that the last $2k$ tapes of M are the oracle tapes. The tapes $k+i$ are oracle query tapes, while tapes $k+2i$ are the respective oracle answer tapes for $i = 1, \dots, k$. We encode the input s as k oracles $\varphi_1, \dots \varphi_k$ defined as follows

$$\varphi_i(0^n) = s_{-n,i}$$
$$\varphi_i(1^n) = s_{n,i}$$

where $\dots s_{-2,i}s_{-1,i}s_{0,i}s_{1,i}s_{2,i}\dots$ is the content of component i of s. Suppose, without loss of generality, that the domain of dependence is equal to the domain of effect of ϕ and that it has the format $\{-m, \dots, 0, \dots, m\}$ for some $m \in \mathbb{N}$ and that no shift of ϕ on a component has absolute value greater that $m/2$. Initially the oracle Turing machine has empty tapes. Then it starts the computation and proceed as follows. First it queries the oracles on tapes $k+i$, for $i = 1, \dots, k$, obtaining $s_{0,i}$ for each (oracle) tape $k+2i$, for $i = 1, \dots, k$ (note that tapes $k+i$ were empty before the query). Then it copies the symbols $s_{0,i}$ to tape i for $i = 1, \dots, k$ and moves the tape head one position to the left on the tapes $i = 1, \dots, k$. Next the oracle Turing machine writes 0 on all the tapes $k+i$, $i = 1, \dots, k$, and consults the oracles, getting $s_{-1,i}$ for each (oracle) tape $k+2i$, for $i = 1, \dots, k$. Then it copies the symbols $s_{-1,i}$ to tape i for $i = 1, \dots, k$ and moves the tape head one position to the left on the tapes $i = 1, \dots, k$. Next the oracle Turing machine appends 0 to the left of the string 0 on tapes $k+i$ (that now have content 00) and queries the oracles, getting $s_{-2,i}$ for each (oracle) tape $k+2i$, for $i = 1, \dots, k$. Then it copies the symbols $s_{-2,i}$ to tape i for $i = 1, \dots, k$ and moves the tape head one position to the left on the tapes $i = 1, \dots, k$. Next the oracle Turing machine appends one more 0 to the left of string 00 on tapes $k+i$ (that now have content 000) and repeats this process until the tapes $k+i$, for $i = 1, \dots, k$, have contents 0^m. At that point the contents of tape i, for $i = 1, \dots, k$, will be $\dots Bs_{-m,i}\dots s_{-2,i}s_{-1,i}s_{0,i}B\dots$. Note that, since m is fixed, this can be done with a finite number of steps.

The next step is to erase the 0's on tapes $k+i$, for $i = 1, \dots, k$ (thus emptying those tapes) and to next write the symbol 1 on those tapes. We also move the tape head of the first k tapes to the position to the right of $s_{0,i}$. Next M queries its oracles, obtaining $s_{1,i}$ on tapes $k+2i$, for $i = 1, \dots, k$. It now copies the symbols $s_{1,i}$ to tape i for $i = 1, \dots, k$ and moves the tape head one position to the right on the tapes $i = 1, \dots, k$. Next the oracle Turing machine appends one 1 at the left of the string 1 on tapes $k+i$ (which now have content 11), for $i = 1, \dots, k$, and repeats the process similarly as done in the case of 0's, obtaining

$$\dots Bs_{-m,i}\dots s_{-2,i}s_{-1,i}s_{0,i}s_{1,i}s_{2,i}\dots s_{m,i}B\dots \tag{20}$$

on tape i for $i = 1, \dots, k$ and deletes the contents of tapes $k+i$, for $i = 1, \dots, k$. We leave the tape heads on the elements $s_{0,i}$ on tapes $i = 1, \dots, k$, i.e. on the center of the domain of dependence/effect. This can be done in time bounded by a finite fixed amount $B_0 > 0$.

The above procedure provides the initial setup for the oracle Turing machine to simulate ϕ. Next, let us see how the oracle Turing machine can simulate one iteration of ϕ. First, ϕ will eventually change the symbols in the domain of dependence/effect. Hence, the oracle Turing machine may need to update the symbols in tapes $i = 1, \dots, k$ corresponding to those elements in the domain of dependence/effect of ϕ, namely the elements given by (20). Since this action depends on only $2m+1$ elements for each component, the oracle Turing machine just needs to read those $2m+1$ elements for tapes $i = 1, \dots, k$, memorizing them on the state, and then decides which should be the new elements which will replace them and then update them. This can be done in time bounded by a finite fixed amount. The tape i of the oracle Turing machine will now have the following contents, where the arrow denotes the position of the tape head

$$\dots BBs'_{-m,i}\dots s'_{-2,i}s'_{-1,i}\overset{\downarrow}{s'_{0,i}}s'_{1,i}s'_{2,i}\dots s'_{m,i}BB\dots \tag{21}$$

Next ϕ will shift each component i by some amount $m_i \le m/2$ for $i = 1, \dots, k$. This can be done by moving the corresponding tape head by m_i positions (if $m_i > 0$, then the head moves m_i positions to the right, otherwise it moves the head $-m_i$ positions to the left). However, a problem becomes apparent for the oracle Turing machine. If $m_i > 0$, then afterwards the tape head will be on position $s_{m_i,i}$ as illustrated below

$$\dots BBs'_{-m,i}\dots s'_{-m+m_i-1}\underbrace{s'_{-m+m_i}\dots s'_{-1,i}s'_{0,i}s'_{1,i}\dots \overset{\downarrow}{s'_{m_i,i}}\dots s'_{m,i}B\dots BB}_{\text{domain of dependence for component } i \text{ of } \phi(s)}\dots$$

and so the oracle Turing machine catches m_i B's, instead of the symbols $s_{m+1,i}\dots\dots s_{m+m_i,i}$ of the component i of s, when trying to compute the next iteration of ϕ. If we do not update the blank symbols to the right of $s'_{m,i}$ to $s_{m+1,i}\dots s_{m+m_i,i}$ (notice that symbols which were previously B's were never previously in the domain of dependence/effect and thus are

equal to the symbols of s at the same positions), then the oracle Turing machine will not be able to update correctly its domain of dependence/effect on the next iteration of ϕ.

We can solve this problem by, after shifting (21), being careful to update the blank symbols now in the domain of dependence/effect of ϕ by the respective symbols $s_{m+1,i} \dots s_{m+m_i,i}$. This can be done by querying the oracle for tape $k+i$, but for this we need to keep track of the position, in component i, of the current index 0 of the current state $\phi(s)$ relative to the index 0 of s. This can be done via the contents of tape $k+i$. Namely, assuming the case of the first iteration of ϕ on s, if $m_i = 0$ we leave the tape $k+i$ empty. If $m_i < 0$, we write $|m_i|$ 0's on tape $k+i$, and if $m_i > 0$, we write m_i 1's on tape $k+i$. In this way, using tape $k+i$, we are able to know for each component $i = 1, \dots, k$ what is the distance of the current position of index 0 of $\phi(s)$ from index 0 of s, i.e. the total amount of shift which was performed relative to the initial state s. If the tape $k+i$ is empty, those indexes coincide (distance $= 0$). If the tape $k+i$ has j 0's (1's), then the index 0 of $\phi(s)$ for component i is j positions to the left (right, respectively) of the index 0 of the same component of s. This encoding can also be used for subsequent iterations of ϕ to register the distance (i.e. total amount of shift), on component i, between the index 0 of $\phi^n(s)$ and the index 0 of s, by appending/deleting 1's, and writing 0's if needed when ϕ shifts several times left. Note that, at each iteration, one just needs to write/delete at most $m/2$ symbols (the bound on all possible shift values) on tape $k+i$ to update this position. To update the blanks on tape i, we just need to proceed as follows. Assume, without loss of generality, that $m_i > 0$ (the case $m_i < 0$ is similar and in the case $m_i = 0$ there is nothing to do). Then, as mentioned above, we write m_i 1's on tape $k+i$ (which was empty) to know that the current index 0 is m_i positions to the right of the index 0 of the component i of s. If there would be subsequent shifts to the left, we would append to the string of 1's a number of 1's equal to the shift size. If a subsequent shift is to the right, then we subtract the corresponding number of 1's and eventually write some zeros if the number of 1's is less than the amount of right shift.

Now that we know which is the position of the index 0 of $\phi(s)$ (or, more generally, of $\phi^n(s)$) relative to the index 0 of s and we know that the last move/shift was by $m_i > 0$ elements, we have to update m_i blank symbols by $s_{m+1,i} \dots s_{m+m_i,i}$. We do this similarly as we have done when constructing the initial setup for the oracle Turing machine. Namely, we append up to m_i 1's to the string of 1's on tape $k+i$, one by one, and with each added 1 we move the tape head of tape i by one position to the right. If the tape head of tape i is reading a blank symbol, then we query the oracle on tape $k+i$, obtaining $s_{m+1,i}$ on tape $k+2i$. We replace the first B to the right of $s'_{m,i}$ in tape i by $s_{m+1,i}$. We then append another 1 to tape $k+i$ and repeat the process until we have appended m_i 1's to the right of the initial string of 1's in tape $k+i$ and thus the contents of the tape i is

$$\dots BBs'_{-m,i} \dots s'_{-m+m_i-1} \underbrace{s'_{-m+m_i} \dots \overset{\downarrow}{s'_{m_i,i}} \dots s'_{m,i} s_{m+1,i} \dots s_{m_i+m,i}}_{\text{domain of dependence for component } i \text{ of } \phi(s)} BB \dots$$

If needed, i.e. if $m_i < 0$, we do a similar procedure to update blank symbols at the left of the non-zero string in (21). This finishes one iteration of ϕ. By repeating this procedure several times, we are able to simulate the iteration of ϕ an arbitrary number of times. □

8. Computation with Cantor piecewise affine maps over the real numbers

In this section we show how we can compute with real numbers using Cantor piecewise affine maps.

To achieve this aim, we start with computation over the real numbers with (multidimensional) generalized shifts.

Below we provide a notion of computable real number for multidimensional generalized shifts which is similar in spirit to Definition 3.2 for computable analysis.

Definition 8.1. Let ϕ be a k-dimensional generalized shift with alphabet A, which has at least six symbols $+, -, 0, 1, \cdot$, and b. Then ϕ computes a real number $x \in \mathbb{R}$ if given some arbitrary $n \in \mathbb{N}_0$ and some $s[n] \in (\mathbb{A}^{\mathbb{Z}})^k$ such that $s_1[n] = \dots (b) b.0^n b(b) \dots$, where

$$0^n = \underbrace{0 \dots 0}_{n \text{ times}}$$

and $s_2[n] = \dots = s_k[n] = \dots (b) b.b(b) \dots$, one has:

1. **Halting condition**: Given any $n \in \mathbb{N}_0$, there is some $j \geq 0$ such that

$$\phi^j_{k-1}(s[n]) = \dots (b) b.0b(b) \dots \tag{22}$$

where $\phi^j_{k-1}(s[n])$ denotes the component $k-1$ of $\phi^j(s[n])$. Furthermore, if (22) holds for some $j \in \mathbb{N}$, then $\phi^i(s[n]) = \phi^j(s[n])$ for all $i \geq j$.

2. **Correct result**: If (22) holds, then

$$\phi_k^j(s[n]) = \dots(b)b.a_0 \dots a_l b(b)\dots \tag{23}$$

where $a_0 \dots a_l \in \{+, -, 0, 1, \cdot\}$ encodes in binary format a number $q = j/2^{-n} \in \mathbb{Q}$, with $j \in \mathbb{Z}$, satisfying

$$|x - q| \leq 2^{-n}. \tag{24}$$

We note that we can also say that ϕ computes a real number $x = (x_1, \dots, x_l) \in \mathbb{R}^l$ if ϕ computes each of the points $x_1, \dots, x_l \in \mathbb{R}$ in l components.

Now that we have a notion of computation for points, we can also define similarly computation of a function $f : \mathbb{R}^n \to \mathbb{R}$ with a generalized shift. However, a crucial element that we do not directly have is an oracle to encode approximations of the input as in Definition 3.4. This can be surpassed by writing down all the elements $\varphi(1), \varphi(2), \dots$ of the oracle φ in a single component, similarly as done for Type-2 machines in [9] and in the proof of Theorem 7.2. For example, a real number $x \in [0, 1]$ can be represented as an oracle φ with the property that $\varphi(n)$ represents in binary form, and with $n + 1$ symbols (one symbol for the sign, plus n symbols for the non-negative part), a number $q \in \mathbb{Q}$ such that (24) holds. Then we can represent φ, and thus x, as a component

$$\dots(b)b.\varphi(1)b\varphi(2)b\varphi(3)\dots \tag{25}$$

Definition 8.2. Let ϕ be a k-dimensional generalized shift ϕ with alphabet A, which has at least six symbols denoted by $+, -, 0, 1, \cdot$, and b. Then ϕ computes a function $f : \mathbb{R} \to \mathbb{R}$ if given some arbitrary $n \in \mathbb{N}_0$, and $x \in \mathbb{R}$, if $s[n, x] \in (\mathbb{A}^{\mathbb{Z}})^k$ is such that $s_1[n, x] = \dots(b)b.0^n b(b)\dots$, $s_2[n, x]$ is given by (25) where the oracle φ represents x as explained before (25), and $s_3[n] = \dots = s_k[n] = \dots(b)b.b(b)\dots$, then the following holds:

1. **Halting condition**: Given any $n \in \mathbb{N}_0$, there is some $j \geq 0$ such that

$$\phi_{k-1}^j(s[n, x]) = \dots(b)b.0b(b)\dots \tag{26}$$

where $\phi_{k-1}^j(s[n])$ denotes the component $k - 1$ of $\phi^j(s[n])$. Furthermore, if (26) holds for some $j \in \mathbb{N}$, then $\phi^i(s[n]) = \phi^j(s[n])$ for all $i \geq j$.
2. **Correct result**: If (26) holds, then

$$\phi_k^j(s[n, x]) = \dots(b)b.a_0 \dots a_l b(b)\dots \tag{27}$$

where $a_0 \dots a_l \in \{+, -, 0, 1, \cdot\}$ encodes in binary format a number $q = j/2^{-n} \in \mathbb{Q}$, with $j \in \mathbb{Z}$ satisfying

$$|f(x) - q| \leq 2^{-n}. \tag{28}$$

Similarly as noted above we can compute a function $f : \mathbb{R}^m \to \mathbb{R}^l$ by using the first m components of $s[n, x]$ to encode each component of $x = (x_1, \dots, x_m)$, and the last l components of ϕ to encode 2^{-n}-approximations of each of the l components of $f(x)$.

Now that we have seen how we can compute over the reals with a generalized shift, we can now show how to compute with Cantor piecewise affine maps. The idea is to use the previous definitions for the generalized shift and adapt then to Cantor piecewise affine maps via the map S^{-1} defined by (10).

Definition 8.3. Let Ψ be a Cantor piecewise affine map defined over $C_{m,l}^{2k}$, where $m \geq 6$, and define a map $S_m^{-1} : A^{\mathbb{Z}} \to C_m$ from $A^{\mathbb{Z}}$, where A is an alphabet which has at least six symbols $+, -, 0, 1, \cdot$, and b by

$$S_m^{-1}((s_i)_{i\in\mathbb{Z}}) = \left(\sum_{i=1}^{\infty} \frac{2s_{-i}}{(2k-1)^i}, \sum_{i=1}^{\infty} \frac{2s_{i-1}}{(2k-1)^i} \right) \tag{29}$$

where is the latter equation s_i is seen as a number in $\{0, 1, \dots, m - 1\}$. Then Ψ computes a real number $x \in \mathbb{R}$ if given some arbitrary $n \in \mathbb{N}_0$ and some $s[n] \in (\mathbb{A}^{\mathbb{Z}})^k$ such that $s_1[n] = \dots(b)b.0^n b(b)\dots$, where

$$0^n = \underbrace{0 \dots 0}_{n \text{ times}}$$

and $s_2[n] = \dots = s_k[n] = \dots(b)b.b(b)\dots$, one has:

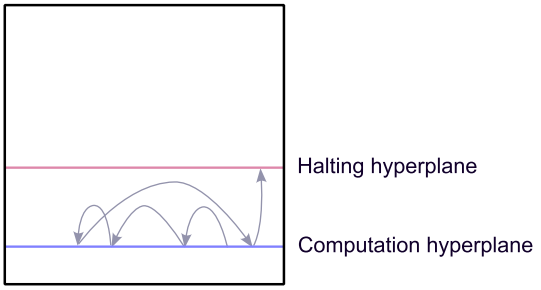

Fig. 3. Graphical illustration of computation with Cantor piecewise affine maps.

1. **Halting condition**: Given any $n \in \mathbb{N}_0$, there is some $j \geq 0$ such that

$$S_m \circ \Psi_{k-1}^j \circ (S_m^{-1})(s[n]) = \dots (b)b.0b(b) \dots \tag{30}$$

where $\Psi_{k-1}^j(x)$ denotes the component $k-1$ of $\Psi^j(x)$. Furthermore, if (30) holds for some $j \in \mathbb{N}$, then $\Psi_{k-1}^i \circ (S_m^{-1})(s[n]) = \Psi_{k-1}^j \circ (S_m^{-1})(s[n])$ for all $i \geq j$.
2. **Correct result**: If (30) holds, then

$$S_m \circ \Psi_k^j \circ (S_m^{-1})(s[n]) = \dots (b)b.a_0 \dots a_l b(b) \dots$$

where $a_0 \dots a_l \in \{+, -, 0, 1, \cdot\}$ encodes in binary format a number $q = j/2^{-n} \in \mathbb{Q}$, with $j \in \mathbb{Z}$, satisfying

$$|x - q| \leq 2^{-n}.$$

Note that condition 1 above says that if the computation starts in an hyperplane given by $x_{2k-2} = 0$, assuming that to the symbol b we associate the number 0 in (29) and we can assume, without loss of generality, that the computation (iteration of Ψ) continues in this hyperplane and then halts when the iteration reaches the hyperplane $x_{2k-2} = 2/(2k-1)$ if we assume that the symbol 0 is encoded into the number 1 in (29). The output is then encoded in the hyperplane x_{2k}. See Fig. 3 for a graphical depiction of this computation process.

We also notice that, using the formalism of computable analysis via encodings, it is possible to define computable sets, etc. Here we just present the notion of computable function, which is based on Definition 8.2.

Definition 8.4. Let Ψ be a Cantor piecewise affine map defined over $C_{m,l}^{2k}$, where $m \geq 6$, and define a map $S_m^{-1} : A^{\mathbb{Z}} \to \mathcal{C}_m$ from $A^{\mathbb{Z}}$, where A is an alphabet which has at least six symbols $+, -, 0, 1, \cdot$, and b by (29). Then Ψ computes a function $f : \mathbb{R} \to \mathbb{R}$ if given some arbitrary $n \in \mathbb{N}_0$, and $x \in \mathbb{R}$, if $s[n, x] \in (\mathbb{A}^{\mathbb{Z}})^k$ is such that $s_1[n, x] = \dots (b)b.0^n b(b) \dots$, $s_2[n, x]$ is given by (25) where the oracle φ represents x as explained before (25), and $s_3[n] = \dots = s_k[n] = \dots (b)b.b(b) \dots$, then the following holds:

1. **Halting condition**: Given any $n \in \mathbb{N}_0$, there is some $j \geq 0$ such that

$$S_m \circ \Psi_{k-1}^j \circ (S_m^{-1})(s[n, x]) = \dots (b)b.0b(b) \dots \tag{31}$$

where $\Psi_{k-1}^j(x)$ denotes the component $k-1$ of $\Psi_{k-1}^j(x)$. Furthermore, if (31) holds for some $j \in \mathbb{N}$, then $\Psi_{k-1}^i \circ (S_m^{-1})(s[n, x]) = \Psi_{k-1}^j \circ (S_m^{-1})(s[n, x])$ for all $i \geq j$.
2. **Correct result**: If (31) holds, then

$$S_m \circ \Psi_k^j \circ (S_m^{-1})(s[n, x]) = \dots (b)b.a_0 \dots a_l b(b) \dots$$

where $a_0 \dots a_l \in \{+, -, 0, 1, \cdot\}$ encodes in binary format a number $q = j/2^{-n} \in \mathbb{Q}$, with $j \in \mathbb{Z}$ satisfying

$$|f(x) - q| \leq 2^{-n}.$$

9. Equivalence of the generalized shift with computable analysis over the real numbers

We now combine the results of the previous sections to show that computation with Cantor piecewise affine maps is equivalent to computable analysis over the real numbers. This result is shown for generalized shifts, which then easily extends to Cantor piecewise affine maps using the map S_m^{-1} of (29) and Theorem 6.5.

9.1. Equivalence at the computability level

We now show that the generalized shift is equivalent to computable analysis over the real numbers at a computability level.

Theorem 9.1. *A real number $x \in \mathbb{R}$ is computable in the computable analysis sense if and only if it is computable by a generalized shift.*

Proof. According to Definition 3.2, a real number $x \in \mathbb{R}$ is computable in the computable analysis sense if it has a computable name. Hence, there is a Turing machine M that, on input $n \in \mathbb{N}$ outputs two integer $p_n, q_n \in \mathbb{Z}$, $q_n > 0$, such that $|x - p_n/q_n| \leq 2^{-n}$. This implies that

$$\left| \frac{p_{n+1}}{q_{n+1}} - x \right| \leq 2^{-(n+1)}. \tag{32}$$

Now notice that there is always some $u_n \in \mathbb{Z}$ such that

$$\left| \frac{p_{n+1}}{q_{n+1}} - \frac{u_n}{2^n} \right| \leq 2^{-(n+1)} \tag{33}$$

which is equivalent to

$$\left| \frac{p_{n+1}2^{n+1}}{q_{n+1}} - 2u_n \right| \leq 1$$

Hence to compute u_n, we just need to do the Euclidean division of $p_{n+1}2^{n+1}$ by q_{n+1} and take the quotient q. We take $u_n = q/2$ if q is even, and $u_n = (q+1)/2$ otherwise. Now (32) and (33) yield

$$\left| \frac{u_n}{2^n} - x \right| \leq 2^{-n}.$$

Notice also that u_n is computable from p_{n+1} and q_{n+1} and thus from n. Moreover, we can compute the binary expansion of u_n from n. This implies that $x \in \mathbb{R}$ is computable if there is a computable function $f : \mathbb{N} \to \{+, -, 0, 1, \cdot\}^*$ such that $f(n) = a_0 \dots a_l \in \{+, -, 0, 1, \cdot\}$ encodes in binary format a number $q \in \mathbb{Q}$ satisfying (24). Hence, from Corollary 7.4, we conclude that the Turing machine computing f is computable by a generalized shift and, due to Definition 8.1, x is also computable by a generalized shift.

Reciprocally, if x is also computable by a generalized shift ϕ, due to Theorem 7.7, ϕ can be simulated by a Turing machine M. Note that, due to Definition 8.1, which does not involve the use of external outputs besides the desired precision of the output, and the proof of Theorem 7.7, there is no need to use an oracle for M. Thus we conclude that M computes a name for x, i.e. that x is computable in the computable analysis sense. □

The proof of the above theorem shows that using a general format for names of a real number x, as done in Definition 3.2, is equivalent to use binary names. However, we have used binary names in Definition 8.1 because they are more "compact", which is useful when studying computational complexity, as we will see in Section 9.2.

We now get the similar result for Cantor piecewise affine maps.

Theorem 9.2. *A real number $x \in \mathbb{R}$ is computable in the computable analysis sense if and only if it is computable by a Cantor piecewise affine map.*

Proof. Use the previous theorem, the map S_m^{-1} of (29) and Theorem 6.5. □

Now let us turn to the case of real functions.

Theorem 9.3. *A function $f : \mathbb{R} \to \mathbb{R}$ is computable in the computable analysis sense if and only if it is computable by a generalized shift.*

Proof. Suppose that $f : \mathbb{R} \to \mathbb{R}$ is computable according to Definition 3.4 and thus is computable by an oracle Turing machine M. Let φ be an oracle that represents x as in (25). Then M with input $n \in \mathbb{N}$ and oracle φ computes $f(x)$ with accuracy 2^{-n}. But M can be simulated by a generalized shift ϕ due to Corollary 7.3 and hence, due to Definition 8.2, the function f is computable by a generalized shift.

Reciprocally, let $f : \mathbb{R} \to \mathbb{R}$ be a function computable by a generalized shift ϕ. By Theorem 7.7, there is an oracle Turing machine N which simulates ϕ. Let $x \in \mathbb{R}$ and let ψ be a name for x. Then proceeding as in the proof of Theorem 9.1, we can convert, in a computable manner, the name ψ to a name φ that is an oracle that represents x as explained before (25).

Then we feed this result, as an oracle, to N, and provide the input n also to N. Then N will compute an approximation of $f(x)$ with accuracy 2^{-n}. This whole procedure (including the conversion of the oracle) defines an oracle Turing machine that, when using oracle ψ, computes $f(x)$. In other words, the oracle Turing machine computes the function f and hence f is computable in the computable analysis sense. □

We now get the similar result with Cantor piecewise affine maps.

Theorem 9.4. *A function $f : \mathbb{R} \to \mathbb{R}$ is computable in the computable analysis sense if and only if it is computable by a Cantor piecewise affine map*

Proof. Use the previous theorem, the map S_m^{-1} of (29) and Theorem 6.5. □

9.2. Equivalence at the computational complexity (polynomial time) level

The above results concern equivalence between computable analysis, the generalized shift, and Cantor piecewise affine maps at a computability level. Let us now show that these models are equivalent at a computational complexity level, namely by showing that the real numbers/functions which are computable in polynomial time in the computable analysis setting are exactly the same as those computable in polynomial time by the generalized shift and by Cantor piecewise affine maps. But to be able to discuss these type of result, we first need to know want is meant by a real number/function computable in polynomial time in these models.

To work with complexity over the real numbers, it is convenient to use a compact representation as a name ψ of a real number $x \in \mathbb{R}$. For example, if $x = \pi$, in Definition 3.2 it makes no difference if $\psi(1) = 3$ or if $\psi(1) = (3^n + 1)/3^n$ since $|3 - \psi(1)| \leq 2^{-1}$ in both cases. However, the time cost of computing the second option can be arbitrarily high, although the computational complexity of the real number π should be intrinsic to itself. One way to solve this problem is to use the binary representation already used in (24), i.e. we assume that $\psi(n)$ encodes in binary format, as a string of $\{+, -, 0, 1, \cdot\}$, a rational number $q = j/2^{-n}$, with $j \in \mathbb{Z}$, such that (24) holds.

Now we can define real numbers and function computable in time $O(t)$ in the computable analysis setting (see [10, Definition 10.1] or [8, p. 47]). We note that, as usual, when using oracle Turing machines we assume that querying the oracle costs only one step of the computation.

Definition 9.5. Let $t : \mathbb{N} \to \mathbb{N}$ be a total function, i.e. which is defined for every element of $\mathbb{N}$.

1. A real number is computable in time $O(t)$ if there is a Turing machine such that, for any $n \in \mathbb{N}$, if the input 0^n is provided to the Turing machine, it will compute in binary format and in time $O(t)$ a rational number $q = j/2^{-n}$, with $j \in \mathbb{Z}$, satisfying (24).
2. A function $f : [0, 1] \to \mathbb{R}$ is computable in time $O(t)$ if there is a oracle Turing machine that given as inputs a name of $x \in [0, 1]$ using the binary representation and a string 0^n will output in binary format and in time $O(t)$ a rational number $q = j/2^{-n}$, with $j \in \mathbb{Z}$, satisfying $|f(x) - q| \leq 2^{-n}$.

We note that, in the above definition, the usage of a compact set such as $[0, 1]$ is crucial when defining complexity for a function over the real numbers. To see this, notice that over a compact set, the successor function $suc : \mathbb{R} \to \mathbb{R}$ given by $suc(x) = x + 1$ is easily shown to be computable in polynomial time. However, using the above definition, the successor would no longer be polynomial time computable over the whole real line $\mathbb{R}$. Indeed suppose, by absurd, that suc was computable in polynomial time over $\mathbb{R}$. Then there would be a polynomial $p : \mathbb{N} \to \mathbb{N}$, a constant $c \in \mathbb{N}$, and an oracle Turing machine such that, for any $n \in \mathbb{N}$ and $x \in \mathbb{R}$ one could compute in time $\leq cp(n) + c$ a rational q satisfying $|q - suc(x)| \leq 2^{-n}$ with the oracle Turing machine. In particular, taking $n = 1$ would yield that the oracle Turing machine would compute a rational q satisfying $|q - suc(x)| \leq 2^{-1}$ in time $\leq cp(n) + c$. But taking $x = 2^{cp(n)+c+2}$ would create a problem since the Turing machine would not have enough time to write its output.

As mentioned in [57], the problem is that the time bound does not depend on the oracle input (i.e. on x) and only depends on the desired precision n. We could add as another parameter the "size" of the input x, when written in binary notation (which would be $\approx \log(|x|)$), but it is mathematically more elegant to use the approach of [57] and use instead the *size of the oracle*. This also has the added advantage (not explored here) that the complexity is independent of the "encoding efficiency" of the oracle when encoding a real number (compare with the example involving π above).

Using the approach of [57], we now define polynomial time complexity in more general terms which avoid the above problem of compact domains.

Definition 9.6. A second-order polynomial with type-1 variable $\mathtt{L}$ (i.e. which takes as arguments a function $\varphi : \mathbb{N} \to \mathbb{N}$) and type-0 variable $\mathtt{n}$ (i.e. which takes as arguments elements of $\mathbb{N}$) is defined inductively as follows:

1. Any element of $\mathbb{N}$ is a second-order polynomial;

2. The variable $\mathtt{n}$ is a second-order polynomial;
3. If P and Q are second-order polynomials, then so are $P+Q$, $P\cdot Q$, and $\mathtt{L}(P)$.

For example

$$P(\mathtt{L})(\mathtt{n}) = 4\mathtt{L}(\mathtt{L}(\mathtt{n}\cdot\mathtt{n}) + \mathtt{n}^2) + \mathtt{L}(\mathtt{n}) + \mathtt{n} + 1$$

is a second-order polynomial. The second-order polynomial maps a function $L:\mathbb{N}\to\mathbb{N}$ and a value $n\in\mathbb{N}$ in the obvious way. For example, if $L(n)=2n$, we get

$$\begin{aligned} P(L)(n) &= 4L(L(n\cdot n)+n^2)+L(n)+n+1 \\ &= 4\cdot 2(2(n\cdot n)+n^2)+2n+n+1 \\ &= 24n^2+3n+1 \end{aligned}$$

With this definition, we can now define computation in polynomial time over $\mathbb{R}$.

Definition 9.7. One can define polynomial time computability in the computable analysis setting as follows:

1. A real number is computable in polynomial time iff it is computable in time $O(p)$ for some polynomial p.
2. A function $f:\mathbb{R}\to\mathbb{R}$ is computable in polynomial time if there is a second order polynomial P and an oracle Turing machine that given as inputs a name ψ of $x\in\mathbb{R}$ using the binary representation and a string 0^n will output in time $P(|\psi|)(n)$ in binary format a rational number $q=j/2^{-n}$, with $j\in\mathbb{Z}$, satisfying $|f(x)-q|\leq 2^{-n}$.

Similarly, we can define computation in polynomial time over $\mathbb{R}$ for generalized shifts and Cantor piecewise affine maps.

Definition 9.8. One can define polynomial time computability for generalized shifts as follows:

1. A real number x is computable in polynomial time by a generalized shift ϕ if ϕ computes x and if there is a polynomial p such that, if $s_1[n]=\ldots(b)b.0^nb(b)\ldots$ initially in Definition 8.1, then the rational number $q=j/2^{-n}$, with $j\in\mathbb{Z}$ satisfying (24) can be computed by ϕ according to Definition 8.1 in $\leq p(n)$ iterations.
2. A function $f:\mathbb{R}\to\mathbb{R}$ is computable in polynomial time by a generalized shift ϕ if ϕ computes f and if there is a second order polynomial P such that, if $s_1[n]=\ldots(b)b.0^nb(b)\ldots$ initially in Definition 8.2 and ψ is a name of $x\in\mathbb{R}$ using the binary representation, then the rational number $q=j/2^{-n}$, with $j\in\mathbb{Z}$ satisfying (28) can be computed by ϕ according to Definition 8.2 in $\leq P(|\psi|)(n)$ iterations.

Definition 9.9. One can define polynomial time computability for Cantor piecewise affine maps as follows:

1. A real number x is computable in polynomial time by a Cantor piecewise affine map Ψ if Ψ computes x and if there is a polynomial p such that, if $s_1[n]=\ldots(b)b.0^nb(b)\ldots$ initially in Definition 8.3, then the rational number $q=j/2^{-n}$, with $j\in\mathbb{Z}$ satisfying (24) can be computed by ϕ according to Definition 8.3 in $\leq p(n)$ iterations.
2. A function $f:\mathbb{R}\to\mathbb{R}$ is computable in polynomial time by a generalized shift Ψ if ϕ computes f and if there is a second order polynomial P such that, if $s_1[n]=\ldots(b)b.0^nb(b)\ldots$ initially in Definition 8.4 and Ψ is a name of $x\in\mathbb{R}$ using the binary representation, then the rational number $q=j/2^{-n}$, with $j\in\mathbb{Z}$ satisfying (28) can be computed by Ψ according to Definition 8.4 in $\leq P(|\psi|)(n)$ iterations.

We now can state the equivalence results. First we start with the case of real numbers.

Theorem 9.10. *A real number $x\in\mathbb{R}$ is computable in polynomial time in the computable analysis sense if and only if it is computable in polynomial time by a generalized shift.*

Proof. Similar to the proof of Theorem 9.1, but now where we take into account the complexity bounds stated in Corollary 7.4 and Theorem 7.7. Note again that, due to Definition 8.1, which does not involve the use of external outputs besides the desired precision of the output, and the proof of Theorem 7.7, there is no need to use an oracle for M. □

Next we consider the case of real functions.

Theorem 9.11. *A function $f:\mathbb{R}\to\mathbb{R}$ is computable in polynomial time in the computable analysis sense if and only if it is computable in polynomial time by a generalized shift.*

Proof. Similar to the proof of Theorem 9.3, but now taking into account the encodings used and the complexity bounds established by Corollary 7.3 and Theorem 7.7. □

Similar results can be obtained for Cantor piecewise affine maps using the map S_m^{-1} of (29) and Theorem 6.5.

Theorem 9.12. *A real number $x \in \mathbb{R}$ is computable in polynomial time in the computable analysis sense if and only if it is computable in polynomial time by a Cantor piecewise affine map.*

Theorem 9.13. *A function $f : \mathbb{R} \to \mathbb{R}$ is computable in polynomial time in the computable analysis sense if and only if it is computable in polynomial time by a Cantor piecewise affine map.*

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by CIDMA (https://ror.org/05pm2mw36) under the Portuguese Foundation for Science and Technology (FCT, https://ror.org/00snfqn58), Grants UID/04106/2025 (https://doi.org/10.54499/UID/04106/2025) and UID/PRR/04106/2025 (https://doi.org/10.54499/UID/PRR/04106/2025).

Data availability

No data was used for the research described in the article.

References

[1] M. Davis, Hilbert's tenth problem is undecidable, Am. Math. Mon. 80 (1973) 233–269.
[2] Y. Matiyasevich, Hilbert's 10th Problem, The MIT Press, 1993.
[3] J. Mycka, J.F. Costa, A new conceptual framework for analog computation, Theor. Comput. Sci. 374 (1–3) (2007) 277–290.
[4] O. Bournez, M.L. Campagnolo, A survey on continuous time computations, in: S. Cooper, B. Löwe, A. Sorbi (Eds.), New Computational Paradigms. Changing Conceptions of What is Computable, Springer-Verlag, New York, 2008, pp. 383–423.
[5] M. Pégny, How to make a meaningful comparison of models: the Church–Turing thesis over the reals, Minds Mach. 26 (1) (2016) 359–388, https://doi.org/10.1007/s11023-016-9407-0.
[6] O. Bournez, A. Pouly, Handbook of computability and complexity in analysis, in: Ch. A Survey on Analog Models of Computation, Springer International Publishing, Cham, 2021, pp. 173–226.
[7] M.B. Pour-El, J.I. Richards, Computability in Analysis and Physics, Springer, 1989.
[8] K.-I. Ko, Complexity Theory of Real Functions, Birkhäuser, 1991.
[9] K. Weihrauch, Computable Analysis: an Introduction, Springer, 2000.
[10] V. Brattka, P. Hertling, K. Weihrauch, A tutorial on computable analysis, in: S.B. Cooper, B. Löwe, A. Sorbi (Eds.), New Computational Paradigms: Changing Conceptions of What is Computable, Springer, 2008, pp. 425–491.
[11] A.M. Turing, On computable numbers, with an application to the entscheidungsproblem, Proc. Lond. Math. Soc. s2-42 (1) (1937) 230–265.
[12] C.E. Shannon, Mathematical theory of the differential analyzer, J. Math. Phys. 20 (1941) 337–354.
[13] M.B. Pour-El, Abstract computability and its relations to the general purpose analog computer, Trans. Am. Math. Soc. 199 (1974) 1–28.
[14] D.S. Graça, J.F. Costa, Analog computers and recursive functions over the reals, J. Complex. 19 (5) (2003) 644–664.
[15] V. Bush, The differential analyzer. A new machine for solving differential equations, J. Franklin Inst. 212 (1931) 447–488.
[16] O. Bournez, M.L. Campagnolo, D.S. Graça, E. Hainry, Polynomial differential equations compute all real computable functions on computable compact intervals, J. Complex. 23 (3) (2007) 317–335.
[17] O. Bournez, D.S. Graça, A. Pouly, Polynomial time corresponds to solutions of polynomial ordinary differential equations of polynomial length, J. ACM 64 (6) (2017) 38:1–38:76.
[18] C. Moore, Recursion theory on the reals and continuous-time computation, Theor. Comput. Sci. 162 (1996) 23–44.
[19] M. Campagnolo, C. Moore, Upper and lower bounds on continuous-time computation, in: I. Antoniou, C. Calude, M. Dinneen (Eds.), Proc. 2nd International Conference on Unconventional Models of Computation - UMC'2K, Springer, 2001, pp. 135–153.
[20] M.L. Campagnolo, C. Moore, J.F. Costa, An analog characterization of the Grzegorczyk hierarchy, J. Complex. 18 (4) (2002) 977–1000.
[21] O. Bournez, E. Hainry, An analog characterization of elementarily computable functions over the real numbers, in: Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP 2004), in: Lecture Notes in Computer Science, vol. 3142, Springer, 2004, pp. 269–280.
[22] O. Bournez, E. Hainry, Real recursive functions and real extentions of recursive functions, in: M. Margenstern (Ed.), Proc. International Conference on Machines, Computations and Universality (MCU 2004), in: Lecture Notes in Computer Science, vol. 3354, 2004, pp. 116–127.
[23] O. Bournez, E. Hainry, Elementarily computable functions over the real numbers and $\mathbb{R}$-sub-recursive functions, Theor. Comput. Sci. 348 (2–3) (2005) 130–147.
[24] O. Bournez, E. Hainry, Recursive analysis characterized as a class of real recursive functions, Fundam. Inform. 74 (4) (2006) 409–433.
[25] V. Brattka, P. Hertling, Feasible real random access machines, J. Complex. 14 (4) (1998) 490–526.
[26] H.T. Siegelmann, Neural Networks and Analog Computation: Beyond the Turing Limit, Birkhäuser, 1999.
[27] M.L. Campagnolo, C. Moore, J.F. Costa, Iteration, inequalities, and differentiability in analog computers, J. Complex. 16 (4) (2000) 642–660.
[28] J. Mycka, J.F. Costa, The $P \neq NP$ conjecture in the context of real and complex analysis, J. Complex. 22 (2) (2006) 287–303, https://doi.org/10.1016/j.jco.2005.07.003.
[29] W.D. Smith, Church's thesis meets the N-body problem, Appl. Math. Comput. 178 (1) (2006) 154–183.

[30] E. Beggs, J. Tucker, Experimental computation of real numbers by newtonian machines, Proc. Royal Soc. A, Math. Phys. Eng. Sci. 463 (2082) (2007) 1541–1561, https://doi.org/10.1098/rspa.2007.1835.
[31] B. Loff, J.F. Costa, J. Mycka, Computability on reals, infinite limits and differential equations, Appl. Math. Comput. 191 (2007) 353–371.
[32] E. Beggs, J.F. Costa, B. Loff, J.V. Tucker, Computational complexity with experiments as oracles, Proc. Royal Soc. A, Math. Phys. Eng. Sci. 464 (2098) (2008) 2777–2801, https://doi.org/10.1098/rspa.2008.0085.
[33] E. Beggs, J.F. Costa, B. Loff, J.V. Tucker, Computational complexity with experiments as oracles. II. Upper bounds, Proc. Royal Soc. A, Math. Phys. Eng. Sci. 465 (2105) (2009) 1453–1465, https://doi.org/10.1098/rspa.2008.0412.
[34] M. Ziegler, Physically-relativized Church-Turing hypotheses: physical foundations of computing and complexity theory of computational physics, Appl. Math. Comput. 215 (4) (2009) 1431–1447, https://doi.org/10.1016/j.amc.2009.04.062.
[35] A. Kawamura, Lipschitz continuous ordinary differential equations are polynomial-space complete, Comput. Complex. 19 (2) (2010) 305–332.
[36] M. Braverman, A. Grigo, C. Rojas, Noise vs computational intractability in dynamics, in: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, ITCS '12, Association for Computing Machinery, 2012, pp. 128–141.
[37] E. Beggs, J.F. Costa, D. Poças, J.V. Tucker, An analogue-digital Church-Turing thesis, Int. J. Found. Comput. Sci. 25 (4) (2014) 373–389, https://doi.org/10.1142/S0129054114400012.
[38] A. Kawamura, H. Ota, C. Rösnick, M. Ziegler, Computational complexity of smooth differential equations, Log. Methods Comput. Sci. 10 (1:6) (2014) 1–15.
[39] M. Braverman, J. Schneider, C. Rojas, Space-bounded Church-Turing thesis and computational tractability of closed systems, Phys. Rev. Lett. 115 (2015) 098701, https://doi.org/10.1103/PhysRevLett.115.098701, https://link.aps.org/doi/10.1103/PhysRevLett.115.098701.
[40] M. Braverman, C. Rojas, J. Schneider, Tight space-noise tradeoffs in computing the ergodic measure, Sb. Math. 208 (12) (2017) 1758, https://doi.org/10.1070/SM8884.
[41] F. Fages, G. Le Guludec, O. Bournez, A. Pouly, Strong Turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs, in: J. Feret, H. Koeppl (Eds.), Computational Methods in Systems Biology, Springer International Publishing, Cham, 2017, pp. 108–127.
[42] D. Poças, J. Zucker, Approximability in the GPAC, Log. Methods Comput. Sci. 15 (3) (2019), https://doi.org/10.23638/LMCS-15(3:24)2019.
[43] M. Blanc, O. Bournez, A characterisation of functions computable in polynomial time and space over the reals with discrete ordinary differential equations: simulation of Turing machines with analytic discrete odes, in: J. Leroux, S. Lombardy, D. Peleg (Eds.), 48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France, in: LIPIcs, vol. 272, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023, pp. 21:1–21:15.
[44] O. Bournez, R. Gozzi, D.S. Graça, A. Pouly, A continuous characterization of PSPACE using polynomial ordinary differential equations, J. Complex. 77 (2023) 101755, https://doi.org/10.1016/j.jco.2023.101755.
[45] R. Gozzi, D. Graça, Characterizing time computational complexity classes with polynomial differential equations, Computability 12 (1) (2023) 23–57, https://doi.org/10.3233/COM-210384.
[46] I. Koswara, G. Pogudin, S. Selivanova, M. Ziegler, Bit-complexity of classical solutions of linear evolutionary systems of partial differential equations, J. Complex. 76 (2023) 101727, https://doi.org/10.1016/j.jco.2022.101727.
[47] M.W. Hirsch, S. Smale, R. Devaney, Differential Equations, Dynamical Systems, and an Introduction to Chaos, Academic Press, 2004.
[48] C. Moore, Unpredictability and undecidability in dynamical systems, Phys. Rev. Lett. 64 (20) (1990) 2354–2357.
[49] C. Moore, Generalized shifts: unpredictability and undecidability in dynamical systems, Nonlinearity 4 (2) (1991) 199–230.
[50] R. Cardona, E. Miranda, D. Peralta-Salas, F. Presas, Constructing Turing complete Euler flows in dimension 3, Proc. Natl. Acad. Sci. USA 118 (19) (2021), https://doi.org/10.1073/pnas.2026818118.
[51] R. Cardona, E. Miranda, D. Peralta-Salas, Turing universality of the incompressible Euler equations and a conjecture of Moore, in: International Mathematics Research NoticesRnab233, 08 2021, https://academic.oup.com/imrn/advance-article-pdf/doi/10.1093/imrn/rnab233/39904547/rnab233.pdf.
[52] R. Cardona, E. Miranda, D. Peralta-Salas, Computability and Beltrami fields in euclidean space, J. Math. Pures Appl. 169 (2023) 50–81, https://doi.org/10.1016/j.matpur.2022.11.007.
[53] R. Cardona, Hydrodynamic and symbolic models of computation with advice, Rev. Mat. Iberoam. 41 (1) (2024) 313–338.
[54] S. Gangloff, A. Herrera, C. Rojas, M. Sablik, Computability of topological entropy: from general systems to transformations on Cantor sets and the interval, Discrete Contin. Dyn. Syst. 40 (7) (2020) 4259–4286, https://doi.org/10.3934/dcds.2020180.
[55] M. Sipser, Introduction to the Theory of Computation, 3rd edition, Cengage Learning, 2012.
[56] P. Odifreddi, Classical Recursion Theory, vol. 1, Elsevier, 1989.
[57] A. Kawamura, S. Cook, Complexity theory for operators in analysis, ACM Trans. Comput. Theory 42 (2) (2012) 5.
[58] B.M. Kapron, S.A. Cook, A new characterization of type-2 feasibility, SIAM J. Comput. 25 (1) (1996) 117–132, https://doi.org/10.1137/S0097539794263452.
[59] E. Neumann, F. Steinberg, Parametrised second-order complexity theory with applications to the study of interval computation, Theor. Comput. Sci. 806 (2020) 281–304, https://doi.org/10.1016/j.tcs.2019.05.009.
[60] J. Cotler, S. Rezchikov, Computational Dynamical Systems, in: 2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, Los Alamitos, CA, USA, 2024, pp. 166–202.