Theoretical Computer Science 132 (1994) 113–128
Elsevier

Computability with low-dimensional dynamical systems*

Pascal Koiran and Michel Cosnard

Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, 46 Allée d'Italie, F-69364 Lyon Cedex 07, France

Max Garzon

Department of Mathematical Sciences, Memphis State University, Memphis, TN 38152, USA

Communicated by A. Salomaa
Received July 1993
Revised September 1993

Abstract

Koiran P., Cosnard M. and Garzon M., Computability with low-dimensional dynamical systems, Theoretical Computer Science 132 (1994) 113–128.

It has been known for a short time that a class of recurrent neural networks has universal computational abilities. These networks can be viewed as iterated piecewise-linear maps in a high-dimensional space. In this paper, we show that similar systems in dimension two are also capable of universal computations. On the contrary, it is necessary to resort to more complex systems (e.g., iterated piecewise-monotone maps) in order to retain this capability in dimension one.

1. Introduction

First-order recurrent neural networks using the saturated-linear output function σ can simulate universal Turing machines [14] in linear time [13, 15] (i.e., the transition function of a Turing machine can be computed in constant time). The global transition function of such a network of d analog units is a piecewise-linear

Correspondence to: P. Koiran, Laboratorie de l'Informatique de Parallélisme, Ecole Normale Supérieure de Lyon, 46 Allée d'Italie, F-69364 Lyon Cedex 07, France. Email: koiran@lip.cns-lyon.fr.

* This work was done during a visit of Max Garzon at LIP sponsored by the Ministère de la Recherche et de la Technologie. Support by the Programme de Recherches Coordonnées C^3 of the CNRS and the Ministère de la Recherche et de la Technologie, and by the Programme Cogniscience (CNRS) is also acknowledged.

SSDI 0304-3975(93)E0159-2

function $F:[0,1]^d \to [0,1]^d$ ($d=1058$ units are sufficient [13]). This result raises the problem of finding the minimum dimension d for which Turing machine simulation by iteration of piecewise-linear functions is possible. In this paper, we show that $d=2$ is sufficient. Our construction can be used to give another proof of the existence of universal neural networks.

Simulation of cellular automata is also compared to simulation of Turing machines, and the computational capabilities of piecewise-linear functions and other natural classes of functions in dimension 1 are investigated. Here, our main results are that one-dimensional piecewise-linear maps cannot perform universal computations, and that two-dimensional piecewise-linear maps cannot simulate arbitrary cellular automata. These negative results are proved using natural, although somewhat restrictive models of universality. Our positive results are all constructive. Finally, a general framework for studying universal machines is proposed. Such a framework could be of great interest for proving more general negative results.

Iterations of piecewise-linear and piecewise-monotone functions on an interval have been extensively studied in the dynamical systems literature, and have exhibited very rich behavior (see for instance [3, 11]). Whether this assertion remains true for their computational behavior seems to us an important open problem. In [9, 10], Moore studies computation-universal dynamical systems. His main results are similar to our Theorem 3.1: a universal Turing machine can be simulated by a "generalized shift" on a two-dimensional Cantor set, this map can be embedded in a smooth map in $\mathbb{R}^2$, and in a smooth flow in $\mathbb{R}^3$.

2. Preliminaries

The definitions and notations used throughout the paper are listed below. I is the unit interval $[0, 1]$. PL_d is the set of piecewise-linear continuous functions on I^d. More precisely, $f: I^d \to I^d$ belongs to PL_d if

- f is continuous,
- there is a sequence $(P_i)_{1 \leqslant i \leqslant p}$ of convex closed polyhedra (of nonempty interior) such that $f_i = f_{|P_i}$ is affine, $I^d = \bigcup_{i=1}^{p} P_i$ and $\mathring{P}_i \cap \mathring{P}_j = \emptyset$ for $i \neq j$.

In the case $d=1$, we also use the notation $I_i = [c_i, c_{i+1}] = P_i$. Let $(a_i, b_i)_{1 \leqslant i \leqslant p}$ be the parameters such that $f_i(x) = a_i x + b_i$ for $x \in [c_i, c_{i+1}]$. $RPL_1 \subset PL_1$ is the set of functions such that the a_i's, b_i's and c_i's are all rational. Piecewise-analytic and piecewise-monotone functions are defined in the same obvious way. If the set of the I_i's is countably infinite rather than finite, we will speak of *countably piecewise*-linear (or analytic, monotone, etc.) functions. In this case, some of the I_i's may be reduced to a single point.

Given a function $f: I^d \to I^d$, a sequence $(x_i)_{0 \leqslant i \leqslant k-1}$ of distinct points is a period-k cycle if $f(x_i) = x_{i+1 \pmod k}$. A point is said to be of period k if it belongs to a period-k cycle.

We consider one-tape one-head Turing machines, with an alphabet A containing a "blank symbol" B, and a finite state set Q. The space of valid tape configurations is the subset $R \subset A^{\mathbb{Z}}$ of configurations with a finite number of non-blank cells. A configuration of the machine is an element of $C = R \times Q \times \mathbb{Z}$ (the third component stands for the position of the read–write head on the tape). $T: C \to C$ is the usual transition function of a Turing machine.

A one-dimensional cellular automaton is also defined by a finite state set Q, and a local transition function $\delta: Q^3 \to Q$. Its configuration space is $C = Q^{\mathbb{Z}}$, and its global transition function $T: C \to C$ is defined by

$$T(x)_i = \delta(x_{i-1}, x_i, x_{i+1}).$$

C is endowed with the standard "Cantor" topology [12], which is the product topology of the discrete topology on Q. Intuitively, this means that a configuration $c' \in C$ is close to $c \in C$ if there is a large $r > 0$ such that $c'(i) = c(i)$ for $i \in [-r, r]$. T is continuous for this topology, and C is homeomorphic to the middle-third Cantor set of I.

This paper deals only with real time simulations, precisely defined as follows.

Definition 2.1. Let T be the transition function of a machine $\mathcal{M}$ (which may be a Turing machine, a cellular automaton, a pushdown automaton, etc.) and let C be its configuration space. A function $f: I^d \to I^d$ simulates $\mathcal{M}$ if there is an f-stable subset $D \subset I^d$ and a bijective function $\phi: C \to D$ such that

$$T = \phi^{-1} \circ f \circ \phi.$$

Intuitively, this means that in order to apply T, one can encode the configuration with ϕ, apply f, and then decode the result with ϕ^{-1}. For cellular automata, we shall make the additional assumption that ϕ is a homeomorphism (ϕ is a homeomorphism if and only if it is continuous, since C is compact). With this assumption, f simulates a cellular automaton if there is a conjugacy between $f_{|D}$ and T. Conjugacy is a natural tool for studying the dynamical properties of cellular automata and neural networks (see for example [2]).

All encoding and decoding schemes presented in this paper are effective, which is a highly desirable property. However, the negative results remain valid even without such an assumption.

A machine $\mathcal{M}_1$ is usually said to simulate another machine $\mathcal{M}_2$ in real time (respectively, linear time, quadratic time, etc) if a computation of $\mathcal{M}_2$ for t time units can be performed by $\mathcal{M}_1$ in time t (respectively, $O(t)$, $O(t^2)$, ...). Since the class of piecewise-linear (respectively, piecewise-analytic, piecewise-monotone) functions is closed under composition, linear time simulation is equivalent to real time simulation.

3. Universality in dimension 2

In this section, we show that two-dimensional piecewise-linear functions are universal.

Theorem 3.1. *An arbitrary Turing machine can be simulated in linear time by a function of PL_2.*

Proof. We simulate pushdown automata with two binary stacks on the alphabet $\{1, 3\}$. It is well-known that these automata can simulate one-tape Turing machines in linear time [6]. The state of an automaton $\mathcal{M}$ and the content of its stacks are encoded in the radix-4 expansion of a point $(x_1, x_2) \in I^2$. The p_1 first digits of x_1 and p_2 first digits of x_2 can encode the state of $\mathcal{M}$ if it has less than $2^{p_1+p_2}$ states (one could take $p_2 = 0$ but this form is more symmetric). In the following, we assume that the set of states of $\mathcal{M}$ is $Q = \{1, 3\}^{p_1} \times \{1, 3\}^{p_2}$. A state $(q_{i,j})_{1 \leq j \leq p_i}$ and a stack $(s_{i,j})_{1 \leq j \leq n_i}$ ($i = 1, 2$, and $s_{i,j} \in \{1, 3\}$) are thus encoded in the real number

$$x_i = \sum_{j=1}^{p_i} q_{i,j}/4^j + \sum_{j=1}^{n_i} s_{i,j}/4^{p+j}.$$

Note that x_i belongs to a kind of "Cantor-like set" (already used in [13]), since the digits 0 and 2 are forbidden. In the following, $\overline{0.a_1 a_2 \ldots a_n}$ denotes a finite radix-4 expansion. Unless otherwise specified, $a_i \in \{1, 3\}$.

The function f simulating $\mathcal{M}$ is affine on each of the $9|Q|$ products $I_{1,l_1} \times I_{2,l_2}$ with

$$I_{i,l_i} = [l_i, l_i + 1/4^{p_i+1}[\quad \text{and} \quad l_i = \overline{0.q_{i,1} \ldots q_{i,p_i} s_{i,1}},$$

or

$$I_{i,l_i} = \{l_i\} \quad \text{and} \quad l_i = \overline{0.q_{i,1} \ldots q_{i,p_i}}.$$

The stack is nonempty in the first case, and empty in the second one. For $x_i \in I_{i,l_i}$, we denote $\Delta x_i = x_i - l_i$. Let us assume that (x_1, x_2) encodes the state and stacks of $\mathcal{M}$ at time t. Let (q_1', q_2') be the next state of $\mathcal{M}$ (determined by the current state (q_1, q_2) and the top-of-stack letters $s_{i,1}$ and $s_{i,2}$). On $I_{1,l_1} \times I_{2,l_2}$, f is such that $f(x_1, x_2) = (x_1', x_2')$ with

$$x_1' = \overline{0.q_{i,1}' \ldots q_{i,p_i}'} + \Delta x_i'$$

and $\Delta x_i'$ defined as follows:

- $\Delta x_i' = 4\Delta x_i$ if stack i is popped,
- $\Delta x_i' = \dfrac{s_{i,1}}{4^{p_i+1}} + \Delta x_i$ if stack i is unchanged,
- $\Delta x_i' = \dfrac{a_i}{4^{p_i+1}} + \dfrac{s_{i,1}}{4^{p_i+2}} + \dfrac{\Delta x_i}{4}$ if a_i is pushed on stack i.

The two last operations can apply to empty stacks, with the convention $s_{i,1}=0$. It is clear that (x'_1, x'_2) encodes the state and stacks of $\mathscr{M}$ at time $t+1$. f is piecewise-linear since the operations applied to the stacks are the same for all the points of a given product $I_{1,l_1} \times I_{2,l_2}$.

In order to complete the proof, we have to extend f outside

$$\mathscr{C} = \bigcup_{l_1, l_2} (I_{1,l_1} \times I_{2,l_2}),$$

to the whole of I^2. This extension cannot interfere with the simulation of $\mathscr{M}$ since only points of $\mathscr{C}$ are used in a computation. There are continuous piecewise-linear extensions of f, since the distance between two distinct products is greater than zero. As a matter of fact, the supremum distance is bounded below by $\min(1/4^{p_1+1}, 1/4^{p_2+1})$. □

The simulation above is naive in the sense that no attempt was made to follow precisely Definition 1.4. This can be done with a few modifications, outlined below. The usual way of simulating a Turing machine $\mathscr{T}$ by a pushdown automaton is to encode the part of the tape on the left of the read–write head in the first stack, and the part on the right in the second stack. The problem with this encoding is that $\phi(c)$ is unchanged if the read–write head and the content of the tape of a configuration c are submitted to the same translation. This is unacceptable since ϕ is supposed to be injective. Let us consider Turing machines on the alphabet $\{1, 3\}$ with the blank symbol $B \equiv 1$. The idea is to use a marker differentiating cell 0 from the other cells. One might for example work with a two-tape machine, and leave the read–write head of the second tape on cell 0. We decided instead to encode a type configuration τ_1 in a new configuration τ_2 in which $\tau_1(i)$ is coded by the 2 symbols $\tau_2(2i)\tau_2(2i+1)$. One may for instance encode 1 by 11, 3 by 33 if $i \neq 0$, and 1 by 13, 3 by 31 if $i=0$. A machine configuration $c \in C$ with the read–write head on cell h is encoded in the point $\phi(c)=(x_1, x_2)$ with

$$x_i = q_i + y_i, \tag{1}$$

$$q_i = \sum_{j=1}^{p_i} \frac{q_{i,j}}{4^j}, \tag{2}$$

$$y_1 = \sum_{j=1}^{+\infty} \frac{\tau_2(2h-j)}{4^{j+p_1}}, \tag{3}$$

$$y_2 = \sum_{j=1}^{+\infty} \frac{\tau_2(2h+j-1)}{4^{j+p_2}} \tag{4}$$

(the $q_{i,j}$'s and p_i's are defined in the proof of Theorem 3.1). With this injective encoding, the simulation is very similar to the one of the theorem.

A simple radix-2 encoding could be used instead of the Cantor set encoding, but f would not be continuous. f would still be continuous with a unary encoding, but then the simulation would not be feasible in linear time.

A rather crude estimate of the complexity of the construction (i.e., the number S of polygons on which f is affine) is as follows. With q states and l letters ($l=2$ up to now), the number of products $I_{1,l_2} \times I_{2,l_2}$ is $q(l+1)^2$. Every rectangular gap between two products can be divided in two triangles, and each triangle can be filled by an affine function. It is not too difficult to see that we end up with $S=7q(l+1)^2$. In the proof of Theorem 3.1, one may in fact use an arbitrary alphabet (for $l=3$, take a radix-6 encoding using the digits 1, 3 and 5 only). One may also work with downward infinite tapes (such as those defined by Eqs. (3) and (4)), in order to suppress the empty stack test. Hence the complexity is reduced to $S=7ql^2$, and moreover, a Turing machine can be simulated without increasing the number of states or symbols. Minsky constructed a universal Turing machine with seven states and four letters [8], which yields $S=784$.

Our construction can be extended so as to give another proof of the result of [13] on the computational universality of analog networks with saturated-linear outputs. The idea is to construct a neural net computing our universal function $f: I^2 \to I^2$. This can be done easily because we need to compute f on $\mathscr{C}$ only. If $\mathscr{R}$ is the set of products $I_{1,l_1} \times I_{2,l_2}$, the transition function of x_i is given by the formula

$$x_i' = \sigma\left[\sum_{\mathscr{P} \in \mathscr{R}} \sigma(a_{i,\mathscr{P}} x_i + b_{i,\mathscr{P}} + \theta_{\mathscr{P}}(x_1, x_2))\right],$$

with $\sigma(x)=x$ if $x \in I$, $\sigma(x)=0$ if $x \leqslant 0$, $\sigma(x)=1$ if $x \geqslant 1$. On each $\mathscr{P}$, $x_i \mapsto a_{i,\mathscr{P}} x_i + b_{i,\mathscr{P}}$ is the affine function defined in the proof above. $\theta_{\mathscr{P}}(x_1, x_2)=0$ if $(x_1, x_2) \in \mathscr{P}$; if (x_1, x_2) belongs to another product $\mathscr{P}'$, $\theta_P(x_1, x_2)$ is equal to a "large" negative value, in order to ensure $\sigma(a_{i,\mathscr{P}} x + b_{i,\mathscr{P}} + \theta_{\mathscr{P}}(x_1, x_2))=0$. This can be done by a net using σ as a "hardlimiter" (i.e., an input fed to σ should always be smaller than 0 or larger than 1), since the distance between two products in $\mathscr{R}$ is greater than zero.

As mentioned in the introduction, the universal neural network construction of [13] can be viewed as a universality result for piecewise-linear functions in dimension d with $d=1058$. This section shows how to reduce d from 1058 to 2. It may be interesting to notice that d can be reduced from 1058 to 92 with little effort as follows. The universal network of [13] is composed of 4 layers, each layer feeding into the next one and the last back into the first. Since the configuration of the machine simulated by the net is encoded in the first layer of 92 neurons, a universal function with $d=92$ can be obtained by composing the functions computed by the four layers.

4. Cellular automata simulation

In this section, we show that unlike Turing machines, cellular automata cannot be simulated by two-dimensional piecewise-linear functions (it is conjectured that the

same is true in any dimension). In order to do this, we need further assume that the encoding function is continuous. This difference between cellular automata and Turing machines is not a mere artifact due to this additional requirement: as shown at the end of this section, the encoding defined by Eqs. (1)–(4) is actually a homeo-morphism.

A few notations: configurations are identified with bi-infinite words on the alphabet Q; uv denotes the concatenation of the (finite or infinite) words u and v whenever it is possible; $0^{-\infty}$ is the word $(u_i)_{i\leqslant 0}$ such that $\forall i,\ u_i=0$; $Q^{+\infty}$ is the set of right-infinite words $(u_i)_{i>0}$.

Theorem 4.1. *There exist cellular automata that cannot be simulated by a function of PL_2 with a continuous encoding.*

Proof. Let $\mathscr{A}$ be a cellular automaton with two distinguished "stable" states, 0 and 1, such that $\forall (q_l, q)\in\{0,1\}^2$, $\forall q_r\in Q$, $\delta(q_l, q, q_r)=q$. The other states are said to be "nonstable". Let $f\in PL_2$ be a function capable of simulating $\mathscr{A}$.

Assume first that

$$\exists i, k\in\mathbb{N},\quad \phi(0^{-\infty}1^k0^{\infty})\in\mathring{P}_i. \tag{5}$$

Since ϕ is continuous, $\exists l\in\mathbb{N}$, $\forall x\in Q^{+\infty}$, $\phi(0^{-\infty}1^k0^lx)\in P_i$. In the remainder of the proof, we always denote: $a=0^{-\infty}1^k0^l$. Since $f_i\circ\phi(ax)=\phi\circ T(ax)$ and $T(ax)\in aQ^{+\infty}$ (by definition of the stable states), $\phi(aQ^{+\infty})$ is stable by f_i. Hence

$$\forall n\geqslant 0,\quad \forall x\in Q^{+\infty},\quad T^n(ax)=\phi^{-1}\circ f_i^n\circ\phi(ax).$$

The sequence $(f_i^n(\phi(ax))_n$ has a finite number of accumulation points because f_i is affine. ϕ^{-1} is continuous, therefore $(T^n(ax))_n$also has a finite number of accumulation points.

Assume now that (5) does not hold. Some ϕ is injective, there must be an edge E with at least 3 distinct configurations of the form $0^{-\infty}1^p0^{+\infty}$ mapped to E by ϕ. At least one of these configurations (say, $0^{-\infty}1^k0^{+\infty}$) is not mapped to an endpoint of E. If E is the edge of a single polygon P_i (i.e., if E lies on one of the sides of I^2), $\phi(aQ^{+\infty})\subset P_i$ for l large enough. It follows from the same argument as in the first case that any sequence of iterates $(T^n(ax))_n$ has a finite number of accumulation points. Let us finally consider the case $E=P_i\cap P_j$ with $i\neq j$. By continuity, $\exists l$, $\phi(aQ^{+\infty})\in P_i\cup P_j$, and $\phi(aQ^{+\infty})$ is stable by f. E is invariant by f_i and f_j since there are more than 2 fixed points on E. Let H_i (respectively H_j) be the half-plane delimited by E containing P_i (respectively P_j). Consider for instance a configuration $ax\in\phi^{-1}(P_i)$. Several cases can be distinguished.

(1) $f_i(H_i)\subset H_i$. In this case, $T^n(ax)=\phi^{-1}\circ f_i^n\circ\phi(ax)$,

(2) $f_i(H_i)=H_j$. In this case, 2 subcases can be distinguished.

(a) $f_j(H_j)\subset H_j$ and $T^{n+1}(ax)=\phi^{-1}\circ f_j^n\circ\phi(ax)$,

(b) $f_j(H_j)=H_i$ and $T^{2n+1}(ax)=\phi^{-1}\circ(f_j\circ f_i)^n\circ\phi(ax)$.

$f_j \circ f_i$ is also affine, therefore in all cases $T^n(ax)$ has a finite number of accumulation points. Let $\mathscr{A}$ be a cellular automaton such that there exists $x_0 \in Q^{+\infty}$ such that $\forall k, l$, $T^n(0^{-\infty}1^k0^l x_0)$ has an infinite number of accumulation points (for instance, the left shift on the nonstable states: $\delta(q_l, q, q_r) = q_r$ if $q \notin \{0, 1\}$). $\mathscr{A}$ cannot be simulated by a function of PL_2. □

The "next configuration" of a cellular automaton with N consecutive active cells can be computed in time $O(N)$ by a Turing machine. Hence it follows from Theorem 3.1 that a cellular automaton with a finite initial configuration can be simulated in quadratic time by a function of PL_2 (a configuration is finite if only a finite number of cells are in a nonstable state). Definition 2.1 is maybe more relevant for cellular automata working on infinite configurations. Such automata may be viewed as models of massively parallel machines, and are strictly more powerful than Turing machines (since they can deal with infinite inputs such as binary expansions of real numbers [5]). For automata working on finite configurations, Theorem 4.1 should be regarded more as a complexity result than a computability result: cellular automata cannot be simulated in linear time by piecewise-linear functions. It is not surprising that cellular automata simulation is slower than Turing machine simulation, since the former model is intrinsically parallel. Note that Theorem 4.1 actually applies if finite configurations only are considered. In order to show this, it suffices to replace the left shift at the end of the proof by a cellular automaton having an orbit of finite configurations with an infinite number of accumulation points. One may take for example a Turing machine enumerating all the finite sequences of two symbols a and b, and simulate it by a cellular automaton.

Let us now show that Eqs. (1)–(4) define an homeomorphism. In order to have a topology to work with, a Turing machine $\mathscr{T}$ is viewed as a cellular automaton with a new state set

$$Q' = \{1, 3\} \cup (\{1, 3\} \times Q).$$

The configuration space of $\mathscr{T}$ is a proper subset $C' \subset C$: at any given time, there must be exactly one cell in a state of the form (a, q). $c(i) = (a, q)$ if and only if the read-write head of $\mathscr{T}$ is on cell i, and $\mathscr{T}$ is in state q. Otherwise, $c(i) \in \{1, 3\}$. Let $c, c' \in C'$ be two configurations such that $c(h) = (a, q)$ and $c'(i) = c(i)$ if $i \in [-|h| - r, |h| + r]$, for some $r \geqslant 0$. Set $\phi(c') = (x'_1, x'_2)$. It holds that $|x'_1 - x_1| \leqslant 1/4^{p_1 + 2r}$ and $|x'_2 - x_2| \leqslant 1/4^{p_2 + 2r + 2}$. ϕ is thus continuous in c since $\|\phi(c') - \phi(c)\|$ can be made as small as desired by taking r large enough. Conversely, for a given (x_1, x_2), set $c = \phi^{-1}(x_1, x_2)$ and $c(h) = (a, q)$. If $|x'_1 - x_1| \leqslant 1/4^{p_1 + 2r}$ and $|x'_2 - x_2| \leqslant 1/4^{p_2 + 2r + 2}$, then $c'(i) = c(i)$ for $i \in [h - r, h + r]$. Hence ϕ^{-1} is also continuous.

5. One-dimensional universality

The main result of this section is that Turing machines cannot be simulated by one-dimensional piecewise-linear functions with the model of Definition 2.1 (one-stack pushdown automata can be simulated by these functions using a straightforward adaptation of Theorem 3.1). However, it is shown that more complicated functions are universal. Cellular automata are again compared to Turing machines.

Theorem 5.1. *There exist Turing machines that cannot be simulated by a function of* PL_1.

The proof of this theorem is based on Lemma 5.2, which gives a dynamical property differentiating PL_1 from PL_2 (and from arbitrary continuous functions on I).

Proof of Theorem 5.1. Let $\mathscr{T}$ be a Turing machine having infinitely many period k cycles for infinitely many values of k. Assume that $\mathscr{T}$ can be simulated by $f: I \to I$. It follows immediately from Definition 2.1 that distinct period-k cycles of $\mathscr{T}$ are mapped by ϕ to distinct period-k cycles of f. Hence f cannot be piecewise-linear according to Lemma 5.2.

$\mathscr{T}$ can be easily constructed as follows. Starting on a blank cell, the read–write head goes to the right until it arrives on another blank symbol. It then goes back to the left, to the first blank symbol, then again on the right, and so on. If there are $k-1$ cells between the two blanks, $\mathscr{T}$ has a period-$2k$ cycle. It actually has infinitely many period-$2k$ cycles (provided that $|A| \geqslant 2$), since all cells can be assigned arbitrary symbols, except those located between the two blanks. □

Lemma 5.2. $f \in PL_1$ *cannot have infinitely many period-k cycles for infinitely many values of k.*

Proof. The result is a straightforward consequence of Lemma 5.3. If this lemma applies for infinitely many k's then by the pigeonhole principle, there are two intervals L_{k_1} and L_{k_2} such that $L_{k_1} \subset L_{k_2}$. This is impossible by definition of the L_k's. □

Lemma 5.3. *Let* $f \in PL_1$ *be a function with infinitely many period-k cycles. There is an interval* L_k *of the form* $L_k = [c, c_i]$ *or* $L_k = [c_i, c]$ *such that*

(a) $f^k_{|L_k} = Id$,

(b) *For any* $k' < k$ *and any subinterval* $L' \subset L_k$, $f^{k'}_{|L'} \neq Id$.

Proof. The set F of period-k points of f has at least one accumulation point x_∞. $f^k(x) = x$ for $x \in F$, and f^k is piecewise-linear. Hence there is an interval J (with x_∞ as endpoint) such that $f^k_{|J} = Id$, and $|F \cap J| = +\infty$. Let $x_0 \in F \cap J$ be such that the orbit of x_0 does not contain any of the c_i's (such a point exists since there are only finitely

many c_i's, and a given c_i can belong to the orbit of at most k period-k points). Let $(\alpha_j)_{0 \leqslant j \leqslant k-1}$ be the sequence such that $f^j(x_0) \in I_{\alpha_j}$. The affine functions $f_{|I\alpha_0}, \ldots, f_{|I\alpha_{k-1}}$ are such that $f_{|I\alpha_{k-1}} \circ f_{|I\alpha_{k-2}} \ldots \circ f_{|I\alpha_0} = \mathrm{Id}$. Hence for any interval K such that

$$\forall j, 0 \leqslant j \leqslant k-1, \quad f^j(K) \subset I_{\alpha_j} \tag{6}$$

it holds that $f^k_{|K} = \mathrm{Id}$. By continuity, for ε small enough, (6) holds for $K = [x_0 - \varepsilon, x_0 + \varepsilon]$. Let K be a maximal interval such that (6) holds and $x_0 \in K$. There exists l such that c_{α_l} is an endpoint of $f^l(K)$ (otherwise, K could be extended without violating the constraints $f^j(K) \subset I_{\alpha_j}$). Set $L_k = f^l(K)$. (a) clearly holds. For $k' < k$ and $x \in L_k$, $f^{k'}(x) = f_{|I\alpha_{k'+l-1}} \circ f_{|I\alpha_{k'+l-2}} \cdots f_{|I\alpha_l} \circ f_{|I\alpha_l}(x)$ (the subscripts are computed modulo k). The right-hand side cannot be equal to Id on an interval $L' \subset L_k$, otherwise it would be equal to Id on the whole of L_k (since the $f_{|I_j}$'s are affine). This is impossible since x_0 would be of period at most k'. □

Note that this result also applies to piecewise-linear discontinuous functions. and so do its consequences Lemma 5.2 and Theorem 5.1. The same is true for piecewise-analytic functions: the interval extension argument works again because of the isolated zeroes property. Another generalization dealing with "interval encodings" is given in theorem 5.5. Interval encodings are more general than those of Definition 2.1 used in the rest of the paper. The idea is simply to give up the requirement that a given configuration should be always coded by the same point. This point can vary during a computation of the machine, and from one computation to another. It should however belong to a fixed interval for a given configuration. These intervals should also be pairwise disjoint, so that an outside observer could tell at any time in which configuration the machine is. It is shown in [7] that interval encodings are useful when working with certain types of recurrent neural networks instead of piecewise-linear functions, but they do not help for PL_1, as we shall now see.

Definition 5.4. $f: I \to I$ simulates a Turing machine $\mathscr{T}$ of transition function T with respect to an interval encoding ϕ if

- ϕ maps a configuration $c \in C$ to an interval $\phi(c) = [a_c, b_c] \subset I$ such that $c \neq c' \Rightarrow \phi(c) \cap \phi(c') = \emptyset$.
- $\forall c \in C$, $f(\phi(c)) \subset \phi(T(c))$.

Theorem 5.5. *There exist Turing machines that cannot be simulated by a function of PL_1 with an interval encoding.*

Proof (*sketch*). Consider a Turing machine $\mathscr{T}$ having infinitely many period-k cycles for infinitely many values of k, like in the proof of Theorem 5.1 Assume that $\mathscr{T}$ can be simulated by $f \in PL_1$ with an interval encoding. For each cycle of $\mathscr{T}$, f has a cycle of intervals (i.e., a sequence of closed intervals $I_1, \ldots, I_k$ such that $f(I_1) \subset I_2$,

$f(I_2) \subset I_3, \ldots, f(I_k) \subset I_1$). By the intermediate value theorem, a cycle intervals contains a cycle of points. We can thus conclude like in the proof of Theorem 5.1. □

However, if we consider countably piecewise-linear functions, the simulation is possible.

Theorem 5.6. *An arbitrary Turing machine can be simulated in linear time by a countably piecewise-linear function.*

Proof. Let $\mathcal{T}$ be a Turing machine, and q, h, r and s integer encodings of its state, the position of its read–write head, the parts of the tape on the left and on the right of its head, respectively. A configuration of $\mathcal{T}$ is encoded in the rational number $x_{q,h,r,s} = 1/(2^q 3^h 5^r 7^s)$. Since all these numbers are distinct, they form a decreasing sequence $(y_i)_{i \in \mathbb{N}}$. Let f be a function affine on $[y_0, 1]$ on each interval $[y_{i+1}, y_i]$, and mapping each y_i to the encoding of the "next configuration". f is piecewise-linear continuous on $]0, 1]$, and simulates $\mathcal{T}$ by definition. f can be extended continuously in 0 by setting $f(0) = 0$, since a configuration with a "large" value of h, r or s has a successor with a large value of h, r or s. □

The next result deals with piecewise-monotone functions. It is in fact a strengthening of Theorem 5.6, since the function constructed in its proof can be taken countably piecewise-linear.

Theorem 5.7. *An arbitrary Turing machine can be simulated in linear time by a continuous piecewise-monotone function.*

Proof. We simulate a Turing machine $\mathcal{T}$ on an alphabet A endowed with an arbitrary total order having B as smallest element, and assume without loss of generality that $\mathcal{T}$ never writes a blank character. All the configurations in which $\mathcal{T}$ is in a given state q and its read–write head reads a given letter l are encoded in a point of the same interval $I_{q,l} = [a_{q,l}, b_{q,l}]$. All these intervals are disjoint, and $I_{q,l} < I_{q,l'}$ (i.e., $b_{q,l} < a_{q,l'}$) if $l < l'$. It will be shown that $\mathcal{T}$ can be simulated by a function f increasing on each $I_{q,l}$. A piecewise-monotone extension of f on I can then be obtained by just "filling the gaps" between the $I_{q,l}$'s.

Our first goal is to construct for each $I_{q,l}$ a total order on the configurations such that whenever c and c' are encoded in the same interval $I_{q,l}$, $c < c' \Rightarrow T(c) < T(c')$. For a given encoding ϕ, f simulates $\mathcal{T}$ if and only if

$$f(\phi(c)) = \phi(T(c)). \tag{7}$$

Hence if ϕ preserves the order on the configurations, f as defined by Eq. (7) will be increasing on $I_{q,l} \cap \phi(C)$. If c and c' are respectively encoded in two distinct intervals $I_{q,l}$ and $I_{q',l'}$, then $c < c' \Leftrightarrow I_{q,l} < I_{q,l'}$. Let us now construct the order on a given

$\phi^{-1}(I_{q,l})$. Let $l(t)$ be the letter read at time t, starting on configuration c at time 0 (therefore $l(0)=l$). For any two distinct configurations c and c', set

$$t_C = \min\{t \geqslant 0;\ l(t) \neq l'(t)\}$$

($t_C \geqslant 1$ by definition of $I_{q,l}$). Two cases can be distinguished.

1. $t_C < +\infty$. In this case, $c < c'$ if $l(t_C) < l'(t_C)$, otherwise $c' < c$.
2. $t_C = +\infty$. Let h and h' be the positions of the read-write head in c and c'.

(a) c and c' are ordered according to the lexicographic order $\prec$ on the tape. This lexicographic order is defined by the following order on the cells

$$h > h+1 > \cdots > h+n > h+n+1 > \cdots > h-1 > \cdots > h-n > \cdots .$$

In other words, the cells on the right of the head are more significant for $\prec$ than those on the left, and a cell is all the more significant as it is closer to the head.

(b) In case of a tie (denoted by $c \simeq c'$), $c < c'$ if and only if $h < h'$.

In order to show that a total order has actually been defined, the only nontrivial property to be checked is transitivity. Let c, c', $c'' \in \phi^{-1}(I_{q,l})$ be three configurations such that $c < c'$ and $c' < c''$. Item 1–2 below deals for instance with the case where $c < c'$ according to item 1 above, and $c' < c''$ according to item 2.

1–1. Let $t'_C = \min\{t \geqslant 0;\ l'(t) \neq l''(t)\}$. If $t'_C < t_C$, then $l(t) = l''(t)$ for $t < t'_C$, and $l(t'_C) = l'(t'_C) < l''(t'_C)$. If $t'_C \geqslant t_C$, then $l(t) = l''(t)$ for $t < t_C$, and $l(t_C) < l'(t_C) \leqslant l''(t_C)$. In both cases, $c < c''$.

1–2. $l''(t) = l'(t) = l(t)$ for $t < t_C$ and $l''(t_C) = l'(t_C) > l(t_C)$, hence $c'' > c$.

2–1. Same argument as 1–2.

2(a)–2(a). $\prec$ is an order, hence it is transitive.

2(a)–2(b). $l(t) = l'(t) = l''(t)$ for $t \geqslant 0$ and $c \prec c' \simeq c''$, hence $c \prec c''$ and $c < c''$.

2(b)–2(a). Same argument as 2(a)–2(b).

2(b)–2(b). $l(t) = l'(t) = l''(t)$ for $t \geqslant 0$, $c \simeq c' \simeq c''$ and $h < h' < h''$, hence $c < c''$.

We now show that $c < c' \Rightarrow T(c) < T(c')$ if c, $c' \in \phi^{-1}(I_{q,l})$. Let us consider again the three cases 1, 2(a) and 2(b).

1. If $t_C > 1$ (*i.e.*, $l(1) = l'(1)$), $T(c)$ and $T'(c)$ are encoded in the same interval $I_{q',l(1)}$. The action of T simply decreases t_C by one, hence $T(c) < T(c')$. If $t_C = 1$, then $l(1) < l'(1)$, $T(c) \in I_{q',l(1)}$ and $T(c') \in I_{q',l'(1)}$. $T(c) < T(c')$ since $I_{q',l'(1)} < I_{q',l'(1)}$.

2. (a) ($T(c)$, $T(c')$ falls under case 2(a), and $T(c) < T(c')$.

(b) The head of $\mathscr{T}$ moves in the same direction for c or c', hence the pair ($T(c)$, $T(c')$) also falls under case 2(b), and $T(c) < T(c')$.

This ordering does not seem a priori constructive, since in order to compare two configurations, we need to observe the behavior of $\mathscr{T}$ for an unbounded period of time. In fact the converse is true, and the location of a configuration c in the ordering can be characterized by two finite, computable sequences $r(c)$, $u(c) \in A^*$. Let $r'(c) = l(0)\,l(1)\cdots l(t_B)$, where t_B is the smallest time such that $l(t_B) = B$ (possibly, $t_B = +\infty$). $r(c)$ is obtained from $r'(c)$ by deleting a letter $l(t)$ if at time t the read-write head is located on a cell that has been scanned before. Note that $r(c)$ is indeed finite

and computable, even if $t_B = +\infty$. $u(c)$ is the sequence of non-blank letters at $t=0$, the cells being ordered in the same manner as for $\prec$. The ordering on C is simply the lexicographic order on $(r(c), u(c), h)$, i.e.,

$$c<c' \Leftrightarrow \begin{cases} r(c)\prec r(c') & \text{or} \\ r(c)=r'(c) & \text{and} \quad u(c)\prec u'(c) \quad \text{or} \\ r(c)=r'(c) & \text{and} \quad u(c)=u'(c) \quad \text{and} \quad h<h'. \end{cases}$$

Here $\prec$ is the lexicographic order on finite sequences. This is due to the fact that if the cells not scanned between $t=0$ and $t=t_B$ are ever scanned, the most significant cells will be scanned first (but determining whether these cells will be actually scanned is undecidable).

It is now easy to define an order-preserving encoding as follows. Let $x, y>0$ be the ranks of $r(c)$ and $u(c)$ in the sequence of words of A^* lexicographically ordered. Take $\phi(c)=(b_{q,l}-a_{q,l})\theta(c)$, with for instance

$$\theta(c)=1-\cfrac{1}{x-\cfrac{1}{3y+\tanh(h)}}.$$

Set $f(\phi(c))=\phi(T(c))$. As mentioned at the beginning of the proof, f simulates $\mathcal{T}$ and is increasing on $I_{q,l}\cap\phi(C)$. There is a continuous increasing extension of f to the whole of $I_{q,l}$ since the points of $\phi(C)$ are isolated. □

For cellular automata, the picture turns out to be quite different again.

Theorem 5.8. *An arbitrary cellular automaton cannot be simulated with a continuous encoding by a countably piecewise-monotone function.*

Proof (*sketch*). Let f be a piecewise-monotone function simulating a cellular automaton with two stable states 0 and 1, like in the proof of Theorem 3.1. The set of configurations for which all cells are in a stable state is not countable, therefore, one of these configurations is mapped by ϕ to the interior $\mathring{J}$ of an interval J on which f is monotone. A half-line of automata can thus be simulated by iterating the monotone function f_J (same argument as for Theorem 4.1). This is not possible for all cellular automata, because a sequence of iterates of f_J has at most one accumulation point. □

The only positive result for cellular automata that we are aware of essentially goes back to Richardson [12], who proved that cellular automata can be simulated by continuous functions on the standard Cantor set K. This is just because the configuration space C is homeomorphic to K (see Section 2). Since K is compact, any

continuous function on K can be continuously extended on I. It follows that cellular automata can be simulated by continuous functions on I.

6. Discussion

In the future, we hope to prove (or maybe disprove) negative results for more general models of universality. A first step in this direction was made in Theorem 5.5. Instead of attacking this problem directly, it might be useful to first tackle the following related problem, which is more precisely defined, and maybe simpler.

> Is it decidable whether the sequence of iterates of a given piecewise-linear function, on a given starting point, reaches a fixed point?

This problem can be precisely stated as follows.

Problem 6.1. Is it decidable to find, given $f \in RPL_1$ and $x \in \mathbb{Q}$, whether there exists $t \in \mathbb{N}$ such that $f^t(x) = f^{t+1}(x)$?

If this problem is undecidable, one could most likely obtain a universality result for RPL_1, since most undecidability results are based on simulations of Turing machines (or other universal devices). Conversely, if this problem is decidable, RPL_1 is not likely to be universal, since in this case there would be an algorithm solving the halting problem for Turing machines (this latter heuristic argument is meaningful only if the end of the "computation" of f is defined by stabilization on a fixed point).

Proving that a function is not universal with very general definitions of encodings and simulations is likely to be quite difficult (this problem is already difficult for discrete machines). In a discrete framework, we propose the following definitions, coherent with the dynamical system point of view of this paper.

Definition 6.2. A machine $\mathcal{M}$ is defined by a total recursive function $\phi_{\mathcal{M}}: \mathbb{N} \to \mathbb{N}$. The *partial* recursive function $f: \mathbb{N}^p \to \mathbb{N}^q$ computed by $\mathcal{M}$ with respect to the *total* recursive functions $\phi_E: \mathbb{N}^p \to \mathbb{N}$ (encoding function) and $\phi_D: \mathbb{N} \to \mathbb{N}^q$ (decoding function) is defined as follows:

$f(n)$ is defined if and only if there exist $t \in \mathbb{N}$ such that

$$\phi_{\mathcal{M}}^t \circ \phi_E(n) = \phi_{\mathcal{M}}^{t+1} \circ \phi_E(n).$$

In this case,

$$f(n) = \phi_D \circ \phi_{\mathcal{M}}^t \circ \phi_E(n).$$

Intuitively, $\phi_{\mathcal{M}}$ is the transition function of $\mathcal{M}$, and $\phi_{\mathcal{M}}^T \circ \phi_E(n)$ is the state of $\mathcal{M}$ after T computation steps on input n.

Definition 6.3. A machine $\mathcal{M}$ is universal if there is a choice of ϕ_E and ϕ_D such that $\mathcal{M}$ computes a universal function $f: \mathbb{N}^2 \to \mathbb{N}$.

Recall that $f: \mathbb{N}^2 \to \mathbb{N}$ is said to be universal if there is a recursive numbering $\phi \mapsto \langle \phi \rangle$ of (partial) recursive functions $\phi: \mathbb{N} \to \mathbb{N}$ such that $f(\langle \phi \rangle, n) = \phi(n)$ [16].

There is a broad agreement on what a universal machine is for discrete models of computation (e.g., Turing machines, register machines, etc). However, it seems that no formal and general definition was given before ours.

Note that it is important to allow only *total* recursive decoding functions in order to avoid a triviality. If the decoding function can be an arbitrary partial recursive function, then any partial recursive function is computable in real time by a linear machine [17].

Our definitions can be adapted to continuous models of computation by taking $\phi_E: \mathbb{N}^p \to \mathbb{R}$, $\phi_{\mathcal{M}}: \mathbb{R} \to \mathbb{R}$, $\phi_D: \mathbb{R} \to \mathbb{N}^q$, and precising what it means that these functions are recursive. This could be done by using one of the several notions of computability over continuous domains that have been studied (see for instance [16, 1, 4]).

It may also be necessary to change the "halting by fixed point" condition of Definition 6.2. This condition is very natural for discrete models of computation, and is also sufficient for PL_2, as shown in Section 3, and for the neural nets considered in [14, 13]. However, it is likely to be too restrictive for other types of machines, e.g., recurrent neural networks using the so-called "standard sigmoid". The possibility of simulating Turing machines with such networks was raised in [14, 13]. A very general criterion would be to check if $\phi_{\mathcal{M}}^t \circ \phi_E(n)$ belongs to a specified recursive set H (the halting set) in order to decide if $\mathcal{M}$ halts at time t. If H is allowed to be an arbitrary recursive set, it can be easily seen that any machine $\mathcal{M}$ remembering the number of computation steps t and its input n during the course of its computation is universal. Hence this halting criterion is too general; a reasonable trade-off is to require H to be a product of intervals.

Acknowledgment

The authors thank the anonymous referee and Professor Jacques Mazoyer for several useful suggestions.

References

[1] L. Blum, M. Shub and S. Smale, On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines, *Bull. Amer. Math. Soc.* **21**(1) (1989) 1–46.

[2] F. Bothelho and M. Garzon, On dynamical properties of neural networks, *Complex Systems* **5**(4), (1991) 401–403.

[3] P. Collet and J.P. Eckmann, Iterated maps on the interval as dynamical systems, *Progress in Physics*, Vol. I (Birkhäuser, Boston, 1980).

[4] M. Garzon and F. Bothelho, Real computation with cellular automata, in: N. Boccara et al., ed., *Proc. Workshop on Cellular Automata and Cooperative Systems*, (Kluwer, Dorchet, MA, 1993) 191–202.
[5] M. Garzon and S.P. Franklin, Neural computability II, in: *Proc. 3rd Int. Joint Conf. on Neural Networks*, Washington, DC, Vol. 1, (1989) 631–637.
[6] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation* (Addison-Wesley, Reading, MA, 1979).
[7] P. Koiran, *Puissance de calcul des réseaux de neurones artificiels*. Ph.D. Thesis, Ecole Normale Supérieure de Lyon, 1993.
[8] M.L. Minsky, *Computation: Finite and Infinite Machines* (Prentice-Hall, Engelwood Cliffs No. 1967).
[9] C. Moore, Unpredictability and undecidability in dynamical systems, *Phys. Rev. Lett.* **64** (20) (1990) 2354–2357.
[10] C. Moore, Generalized shifts: unpredictability and undecidability in dynamical systems, *Nonlinearity*, **4**, (1991), 199–230.
[11] C. Preston, *Iterates of Piecewise Monotone Mappings on an Interval, Vol. 1347*, Lecture Notes in Mathematics (Springer, Berlin, 1988).
[12] D. Richardson, Tessellation with local transformations, *J. Comput. System Sci.* **6** (1972) 373–388.
[13] H.T. Siegelmann and E.D. Sontag, On the computational power of neural nets, SYCON Report 91-11, Rutgers University, 1991.
[14] H.T. Siegelmann and E.D. Sontag, Turing computation with neural nets, *Appl. Math. Lett.*, **4**(6), (1991) 77–80.
[15] H.T. Siegelmann and E.D. Sontag, On the computational power of neural nets, in: *Proc. 5th ACM Workshop on Computational Learning Theory*, 1992.
[16] K. Weihrauch, *Computability*. EATCS Monographs on Theoretical Computer Science (Springer, Berlin, 1987).
[17]. E.D. Zeigler, Every discrete input machine is linearly simulatable, *J. Comput. System Sci.* (1973) 161–167.