ELSEVIER

Theoretical Computer Science 168 (1996) 461–472

Theoretical Computer Science

The simple dynamics of super Turing theories

Hava T. Siegelmann *

Information Systems Engineering, Faculty of Industrial Engineering and Management, Technion, Haifa 32000, Israel

Abstract

This paper reasons about the need to seek for particular kinds of models of computation that imply stronger computability than the classical models. A possible such model, constituting a chaotic dynamical system, is presented. This system, which we term as the analog shift map, when viewed as a computational model has super-Turing power and is equivalent to neural networks and the class of analog machines. This map may be appropriate to describe idealized physical phenomena.

1. Introduction

A straightforward method of measuring the area of a surface is by counting the number of atoms there. One may be able to develop smart algorithms to group atoms together in sets, and thus speed up the counting time. A totally different approach is by assuming continuous rather than quantized/discretized universe and calculating the relevant integral. Such a continuous algorithm should ideally be implemented on an analog machine, but it can also be approximated by a digital computer that allows for finite precision only. Although the actual hardware is discrete, the core assumption of continuity allows the development of inherently different algorithms to evaluate areas. It is possible that in the theory of computation, we are still at the stage of developing algorithms to count faster. Maybe just by assuming an analog media (although not really having it), we would be able to do much better for some tasks.

A more fundamental reason to look for analog computation models stems from recent advances in the field of physics and the aim to simulate idealized physical phenomena on computers. Already in the 18th century, Poincaré realized that the orbits of simple dynamical systems may be extremely unpredictable, and mathematicians have been dealing with this phenomenon since. However, since 1975 "chaos" has been realized by physicists to occur in many systems of scientific interest [7]. Turing machines are indeed able to simulate a large class of systems, but seem not to capture the whole

* E-mail: iehava@ie.technion.ac.il.

PII S0304-3975(96)00087-4

picture of computation in nature, like the evolvement of many chaotic systems along with their exponential sensitivity to initial values.

We propose an alternative model of computation, whose computational power can surpass that of the Turing model. The proposed model builds on a particular analog chaotic system [6]; by applying the system to computer science, a super-Turing model is developed. (In this paper, the term "super-Turing" is meant to denote any system of computing that incorporates, but is more powerful than, the standard Turing model.) This model assumes analog medium and it demonstrates exponential sensitivity; thus it can be considered as an analog computational paradigm.

2. Analog computation

The term computation is not totally agreed upon. There are those that understand it in terms of standard digital computers and others that interpret computation by means of neuro-modeling of the brain. In the field of analog computation, any experiment done in a physics laboratory is referred to as a computation. The basic characteristic of analog computation that differentiates it from the classical, digital computers is the use of real constants. Physical dynamics can be characterized by the existence of real constants that influence the macroscopic behavior of the system. For example, planetary motion is used to measure time with very high precision although we know the gravitational constant G only to 2 digits. The planets, of course, evolve according to the exact value of G, irrespective of its measurement by humans. The constants have their definite meaning even without being measured, just as the case with the Planck's constant, the charge of the electron, and so forth. Other real values that may affect a laboratory system can be length and mass. (In contrast, in digital computation all constants must be fully known in all their finite digits to the programmer.)

Many chaotic dynamical systems require exact precision of their parameters. Consider for example the Henon map [7], defined by

$$x_{n+1} = a + by_n - x_n^2,$$
$$y_{n+1} = x_n,$$

for constants a and b. The behavior of this system is very sensitive to the choice of its constants. For $a = 1.3$ and $b = 0.3$, the system cycles in a 7-period cycle. When the constant a is minutely increased, the system moves into a 14-period cycle, then into a 28-period cycle, etc. For a further small increase in a, the system gets into a chaotic motion; see Fig. 1 (Fig. 1 was plotted using *Dynamics* [15]). See e.g. [2, 7, 16] for many other systems with similar sensitivity. As for the real constants the dynamics is defined on continuous – rather than discrete – space.

The first to recognize the need of real constants were Blum, Shub and Smale [5]. They suggested a new type of computational model that adheres to the fact that physical systems do not evolve according to the binary representation of their constants

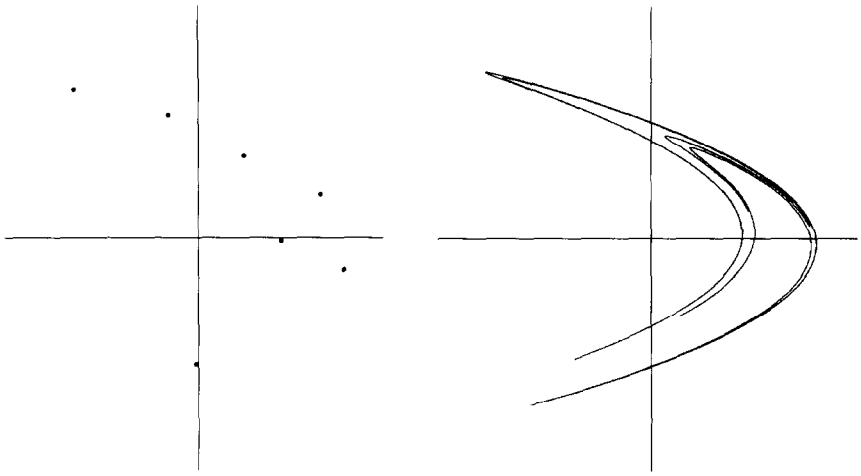

Fig. 1. The Henon map for different constants.

but rather by the values themselves. They call their model "model for computation over the real numbers" [5]. It consists of finitely many building blocks in a recurrent interconnection, where each block computes either a polynomial or a binary decision.

In [22, 23], Siegelmann and Sontag introduced another model of analog computation that is based on systems used in the field of neural networks. In addition to the use of real constants their model is characterized also both by "continuity in each computation step" (i.e., there are no "tests for zero" and non-continuous binary decisions) and by finite dimensionality. Both analog models – the one in [5] and the one in [22] – compute more than the Turing machine does and include some non-recursive functions. In what follows we will refer to the neural networks based model as ARNN (analog recurrent neural networks). The computational power of the ARNN model is currently fully known and it will serve as the basic model in this text.

3. The ARNN model

In the science of computing, machines are classified according to the classes of tasks they can execute or the functions they can compute. The most popular model is the Turing machine, but there are others that result in stronger, though non-realizable, models. "Nonuniform Turing machines" exemplify such models [3]: the machine receives on its tape, in addition to its input, another sequence w_n to assist in the computation. For all possible inputs of the same length n, the machine receives the same advice sequence w_n, but different advice is provided for input sequences of different lengths. We will focus on the class of non-uniform machines that compute in polynomial time (and use a polynomial long advice), denoted by P/poly [3]. The class P/poly strictly includes P and it also computes functions which are non-recursive – "super-Turing" functions. To get an intuition for this class, note that if both advice and time are

exponentially long (i.e., $O(2^n)$), the advice can be used to indicate the desired response for each of the 2^n possible input strings of length n, and, thus, compute all functions $f: \{0,1\}^* \mapsto \{0,1\}$, including non-computable ones. Here in P/poly only polynomial many bits can be used in the advice and only polynomial time is allowed; thus it computes non-recursive functions, but yet only an exponentially small subset of them.

In [22, 23], Siegelmann and Sontag noticed that the non-uniform classes are indeed natural for analog computation models. They introduced the model of computation which is uniform (though includes real constants) but yet has non-uniform super-Turing capabilities; this model is the classical analog recurrent neural network (ARNN), which is popular in practice as a machine having automatic learning and adaptation capabilities [9]. The ARNN consists of a finite number of neurons. The activation of each neuron is updated by the equation

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij}x_j(t) + \sum_{j=1}^{M} b_{ij}u_j(t) + c_i\right), \quad i = 1,\ldots,N, \tag{1}$$

where N is the fixed number of neurons, M is the number of external input signals, x_j are the activations of the neurons, u_j are the external inputs, and a_{ij}, b_{ij}, c_i are the real coefficients, also called constants or weights (the name "analog" is due to the real rather than rational coefficients). The function σ is the simplest possible "sigmoid", namely the saturated-linear function:

$$\sigma(x) := \begin{cases} 0 & \text{if } x < 0, \\ x & \text{if } 0 \leqslant x \leqslant 1, \\ 1 & \text{if } x > 1. \end{cases} \tag{2}$$

A subset of the N neurons is singled out to communicate the output of the network to the environment. Inputs and outputs are streams of letters, and computability is defined under the convention that is sometimes used in practical communication networks: there are two binary input channels, where one is used to carry the binary input signal, and the other one indicates when the input is active. A similar convention is applied to the output.

The ARNN computes the super-Turing class P/poly in polynomial time, and all binary functions in exponential time [22]. This fact is connected to classical computability by observing that when the real weights are constrained to be rational numbers, the network has the Turing power [21, 23]. (Follow-up generalizations appear in [4, 12, 18].) This result implies that the use of real constants in the field of analog computation is closely related to the polynomial nonuniformity in classical digital computation theory.

The ARNN model was suggested as a basic analog computation model, stating that "any reasonable analog computer with the characteristics of real numbers, finite dimensionality, and continuous computation, will have no more power (up to polynomial time) than the analog recurrent networks." The same statement holds for stochastic ARNN [18].

4. The analog shift map

We next present a chaotic dynamical system that computationally is as strong as the ARNN; In the literature of dynamical systems, chaos is commonly exemplified by the "shift map" (such as Baker's map [1] or the Horseshoe map [8]) over a set of bi-infinite dotted sequences. Assume E is a finite alphabet. A dotted sequence over E (denoted by $\dot{E}$) is a sequence of letters where exactly one is the dot sign "$\cdot$" and the rest are all in E. The dotted sequences can be finite, (one-side) infinite, or bi-infinite over E.

Let $k \in \mathbb{N}$ be an integer, the shift map

$$S^k : \dot{E} \to \dot{E} : (a)_i \mapsto (a)_{i+k}$$

shifts the dot k number of places, where negative values cause a left shift and positive ones a right shift. For example,

$$S^3(\cdots a_{-2}\, a_{-1} \,.\, a_1\, a_2\, a_3\, a_4\, a_5 \cdots) = \cdots a_{-2}\, a_{-1}\, a_1\, a_2\, a_3 . a_4\, a_5 \cdots .$$

The "generalized shift" map is defined by Moore [13, 14] as follows: a finite dotted substring is replaced with another dotted substring according to a function G, then this new sequence is shifted an integer number of places either left or right according to a function F. Formally, the generalized shift is the function

$$\Phi : a \mapsto s^{F(a)}(a \oplus G(a)), \tag{3}$$

where the function $F : \dot{E} \to \mathbb{Z}$ indicates the amount of shift (where negative values cause a left shift and positive ones a right shift), and the function $G : \dot{E} \to \dot{E}$ describes the modification of the sequence. Both F and G have a finite *domain of dependence* (DoD), that is, F and G depend only on a finite dotted substring of the sequence on which they act. G has a finite *domain of effect* (DoE), i.e. every sequence in the image of G consists of a finite dotted sequence, padded to both sides by infinitely many ε's, where ε is the "empty element", not contained in E. Note that the DoD and DoE of G do not need to have equal length. Finally, the operation $\oplus$ is defined by

$$(a \oplus g)_i = \begin{cases} g_i & \text{if } g_i \in E, \\ a_i & \text{if } g_i = \varepsilon. \end{cases}$$

The generalized-shift function is homeomorphic to the action of a piecewise differentiable map on a square Cantor set. Moore conjectured that such maps arise in physical systems consisting of a free particle moving between plane mirrors. Most interestingly for the present discussion, Moore proved that the generalized-shift map is computationally equivalent to the Turing machine. This result, thus, connects chaotic dynamical systems with the classical computational model.

Here, we introduce a new chaotic dynamical system: the "analog shift map". It is similar to the generalized-shift function in Eq. (3), except for allowing the substituting

dotted sequence (DoE) defined by G to be finite, infinite, or bi-infinite, rather than finite only. The name "analog shift map" implies the combination of the shift operation with the computational equivalence to the analog computational models (of the ARNN form), as will be proved later.

Example 4.1. To illustrate, assume the analog shift defined by:

DoD	F	G
0.0	1	$\bar{\pi}.$
0.1	1	.10
1.0	0	1.0
1.1	1	.0

Here $\bar{\pi}$ denotes the left infinite string $\cdots 51413$ in base 2 rather than base 10. The table is a short description of the dynamical system, in which the next step depends upon the first letter to the right of the dot and the one to its left. If these letters (i.e. the DoD) are 0.0, then (according to the first row of the table) the left side of the dot is substituted by $\bar{\pi}$ and the dot moves one place to the right. If the DoD is instead 0.1 (as in the second row of the table), the second letter to the right of the dot becomes 0 and there is a right shift; etc.

The dynamic evolving from

000001.10110

is as follows: here the DoD is 1.1, hence (by the fourth row of the table) the letter to the right of the dot becomes 0 and the dot is shifted right:

1.1: (000001.00110) 0000010.011

Now, the DoD is 0.0:

0.0: ($\bar{\pi}$.0110) $\bar{\pi}$0.110

0.1: ($\bar{\pi}$0.100) $\bar{\pi}$01.00

1.0: ($\bar{\pi}$1.00) $\bar{\pi}$01.00

Here the DoD is 1.0 and no changes occur, this is a fixed point.

The computation associated with the analog shift systems is the evolution of the initial dotted sequence until reaching a fixed point, from which the system does not evolve anymore. The computation does not always end; when it does, the input–output map is defined as the transformation from the initial dotted sequence to the final subsequence to the right of the dot. (In the above example, a fixed point is reached in four steps, and the computation was from "000001.10110" to "00".) To comply with computational constraints of finite input/output, attention is constrained to systems that start with finite dotted sequences and halt with either finite or left infinite dotted sequences only. Even under these finiteness constraints, the analog shift computes richer maps than the Turing machines; this will be proved in the next section.

5. The analog shift map is super-Turing

Here we prove the computational equivalence between the analog shift map and the analog recurrent neural network model. Denote by $\mathrm{AS}(k)$ the class of functions computed by the AS map in time k, by $\mathrm{NN}(k)$ the class of functions computed by the ARNN in time k, and by $\mathrm{Poly}(k)$ the class of polynomials in k. The main theorem states that:

Theorem 1. *Let F be a function so that $F(n) \geqslant n$. Then,* $\mathrm{AS}(F(n)) \subseteq \mathrm{NN}(\mathrm{Poly}(F(n)))$ *and* $\mathrm{NN}(F(n)) \subseteq \mathrm{AS}(\mathrm{Poly}(F(n)))$.

Proof. We assume, without loss of generality, that the finite alphabet E is binary; $E = \{0, 1\}$.

1. $\mathrm{AS}(F(n)) \subseteq \mathrm{NN}(\mathrm{Poly}(F(n)))$: Given a bi-infinite binary sequence

$$S = \cdots a_{-3}\, a_{-2}\, a_{-1} \,.\, a_1\, a_2\, a_3 \cdots,$$

we map it into the two infinite sequences

$$S_r = .\, a_1\, a_2\, a_3 \cdots \qquad S_l = .\, a_{-1}\, a_{-2}\, a_{-3} \cdots.$$

A step of the AS map can be redefined in terms of the two infinite sequences S_l and S_r rather than the bi-infinite sequence S itself:

$$\tilde{\phi}(S_l, S_r) = (S_l \oplus G_l(d_l, d_r), S_r \oplus G_r(d_l, d_r)).$$

Here, $d_l = a_{-1}\, a_{-2} \cdots a_{-d}$ and $d_r = a_1\, a_n \cdots a_d$, assuming, without loss of generality, that the DoD is of length $2d$ and is symmetric; that is $|d_l| = |d_r| = d$. The DoE of the binary sequences $G_l(d_l, d_r)$ and $G_r(d_l, d_r)$ may be unbounded.

We next prove the computational inclusion of the analog shift map in neural networks. We do so by implicitly constructing an analog recurrent net that simulates the AS map. This is done using the high-level programming language NIL [19] that is associated with a neural compilation scheme; the compiler translates a NIL program into a network which computes exactly the same.

In the following algorithm, we consider the binary sequences S_l and S_r as unbounded binary stacks; we add two other binary stacks of bounded length, T_l and T_r, as a temporary storage. The stack operations we use are: **Top**(stack), which returns the top element of the stack; **Pop**(stack), which removes the top element of the stack; and **Push**(element, stack), which inserts an element on the top of the stack. We are now ready to describe the parts of the algorithm:

(a) Read the first d elements of both S_l and S_r into T_l and T_r, respectively, and remove them from S_l, S_r.

Procedure Read;
Begin
For $i = 1$ to d
Parbegin

$T_l = \text{Push}\ (\text{Top}(S_l), T_l),\ S_l = \text{Pop}\ (S_l);$
$T_r = \text{Push}\ (\text{Top}(S_r), T_r),\ S_r = \text{Pop}\ (S_r);$
Parend
End;

The same task could be shortly written as the sequence $\mathbf{Push}^d(S_l, T_l)$, $\mathbf{Push}^d(S_r, T_r)$, $\mathbf{Pop}^d(S_l)$, $\mathbf{Pop}^d(S_r)$, where the Push and Pop operations are well defined for any finite d, to be executed d times.

(b) For each choice of the pair

$$v_i = (\xi_l^i, \xi_r^i) \in \{0,1\}^d \times \{0,1\}^d, \quad i = 1, \ldots, 2^{2d}$$

of the DoD, there is an associated pair of substituting strings:

$$(\mu_l^i, \mu_r^i) \in \{0,1\}^{\kappa_l^i} \times \{0,1\}^{\kappa_r^i}$$

(of the DoE), where each length κ_v^i ($i = 1, \ldots, 2^{2d}$, $v \in \{l, r\}$) is either bounded by some constant k or is ∞. We also consider μ's as stacks.

(c) Computing the $\oplus$ function.

Procedure Substitute(μ_l, μ_r, S_l, S_r);
Begin
Parbegin
If($\kappa_l > k$) (* μ_l is Infinitely long *)
then $S_l = \mu_l$
else $S_l = \mathbf{Push}^{\kappa_l}(\mu_l, S_l)$;
If($\kappa_r > k$) (* The parallel case of r *)
then $S_r = \mu_r$
else $S_v = \mathbf{Push}^{\kappa_r}(\mu_r, S_r)$;
Parend
End;

The following program simulates one step of the AS map:

Program AS-step( );
Begin;
Read;
Substitute
End;

That is, there is an ARNN which computes as the AS map, and it is constructible by the methods in [19] from the algorithm above.

2. $\text{NN}(F(n)) \subseteq \text{AS}(\text{poly}(F(n)))$: We next show how to simulate a Turing machine with polynomial advice via an AS map. Because $\text{NN}(\text{poly}) = \text{P/poly}$ this will prove the result. We will use the following observation. Denote the infinite string which is the concatenation of all advice by

$$w = \langle \ldots, w_3, w_2, w_1 \rangle,$$

and the concatenation of the first n advises by

$$w'_n = \langle w_n, \ldots, w_2, w_1 \rangle.$$

Constrained by polynomial computation time it is easy to verify that a Turing machine that receives the polynomially long advice w'_n is equivalent to a Turing machine that receives the advice w. (One side, the $\langle x, w \rangle$ machine retrieves the relevant part and ignores the rest of the information; the other side, the machine $\langle x, w \rangle$ cannot read more than the first polynomially many advice bits during polynomial time computation.)

We now show how to simulate a Turing machine with a polynomial advice via an AS map; our simulation is similar to the one made by Moore, but some preprocessing is required. A configuration of the Turing machine consists of its tape, the relative location of the read/write head in the tape, and its internal control state. Moore encoded the configuration in the bi-infinite string using the fields:

$$\boxed{\bar{0}}\ \boxed{\text{tape-left}} \cdot \boxed{\text{state}}\ \boxed{\text{tape-right}}\ \boxed{\bar{0}}$$

That is, the string starts with infinitely many 0's ($\bar{0}$), followed by the part of the tape to the left of the read/write head, then the decimal point, the internal state of the machine, the part of the tape under the head and to its right, and again infinitely many 0's that encode the empty part of the tape. In each step of Moore's simulation, the DoD contains the state, the tape letter under the read/write head, and the two letters surrounding it. The DoE is such that the operation may simulate writing a new tape letter, entering a new control state, and moving the head one bit to either right or left.

In our simulation, we allow for more flexible encoding of the Turing machine configuration:

$$\boxed{\text{garbage}}\ \boxed{\text{left-end marker}}\ \boxed{\text{tape-left}} \cdot \boxed{\text{state}}\ \boxed{\text{tape-right}}\ \boxed{\bar{0}}$$

We substitute the $\bar{0}$ string to the left of the tape with two fields: garbage and left-end marker. Here, garbage means infinitely many bits with no relevant meaning, and the left-end marker implies that there is no relevant information to its left.

We suggest the following encoding that will allow to clearly interpret the string: 10 will present the 0 letter on the tape, 11 will present the letter 1 on the tape; 01 will present the left-end marker. The internal state of the machine will be encoded by a sequence of 0's only and will end with 1; and $\bar{0}$ still denotes the infinite sequence of 0's that represents the empty right part of the tape.

Assume the Turing machine has the set of internal states $\{p_1, \ldots, p_Q\}$. We add the dummy states $\{q_1, \ldots, q_r\}$ for a constant r. Now the Turing machine with a polynomial advice will act as follows:

(a) The initial bi-infinite string is

$$\boxed{\bar{0}} \cdot \boxed{q_1}\ \boxed{x}\ \boxed{\bar{0}}$$

where x is the input string. (Note that the string is describable in finite terms as required by a computational model.)

(b) In the next step the string is transferred to

$$\boxed{w} \cdot \boxed{q_2} \boxed{x} \boxed{\bar{0}}$$

where $w = \langle \ldots, w_3, w_2, w_1 \rangle$. (As was previously observed, the string w adds no more advise information than the string w'_n for computations on the length n input string.)

(c) In polynomially many steps, the part $\langle w_{n-1} \ldots w_1 \rangle$ is removed, and the remaining part of the infinite string w is partitioned into the relevant part w_n and the garbage part $\langle \ldots w_{n+2} w_{n+1} \rangle$.

$$\boxed{\text{garbage}} \boxed{\text{left-end marker}} \boxed{w_n} \cdot \boxed{q_r} \boxed{x} \boxed{\bar{0}}$$

This is done by a recursive algorithm, that requires linear time in the length of the input string x, and thus can be executed easily by a Turing machine or, equivalently, by a GS map. The w_n part is next transferred to the right side of the tape:

$$\boxed{\text{garbage}} \boxed{\text{left-end marker}} \cdot \boxed{p_1} \boxed{x} \boxed{w_n} \boxed{\bar{0}}$$

where p_1 is the initial state of the Turing machine with advice.

(d) From now on, each step of the machine is simulated as in Moore, with the only one difference that the left-end marker should be preserved to the immediate left of the relevant tape information. □

6. The physical plausibility

The appeal of the analog shift map is not only as an almost classical, chaotic dynamical system that is associated with analog computation models. It is also a mathematical formulation that is conjectured to describe idealized physical phenomena. The idealization allows for model assumptions such as any convenient scale to describe the system, noise-free environment, and physics of continuous medium. Some of the physical models, previously believed to equate Turing machines, turn out to be as strong as the analog models (both to simulate and be simulated by) when their "real nature" is recognized. This assertion can be demonstrated, for example, with the system introduced by Moore [13].

The system is a "toy model", describing the motion of a particle in a three-dimensional potential, such as a billiard ball or a particle bouncing among parabolic mirrors. A finite number of mirrors suffices to describe the full dynamics, one mirror for each choice of the DoD. The (x, y) coordinates of the particle, when passing through a fixed, imaginary plane $[0, 1] \times [0, 1]$, simulate the dotted sequence "$x.y$". To define the computation, the particle starts in input location $\bar{0}.y$ where y is the finite input string; the output is defined in finite terms as well. The main difference between Moore's view and ours is that, for us, the characterizations of the few mirrors cannot

be fully described finitely, although we are not necessarily interested in more than some precision to some computation. On one side, analog shift can simulate this system, even with unbounded characteristics. To simulate P/poly, we note that the advice can be encoded in a uniform manner by the characterizations of the mirrors (e.g., the concatenation of all advice can be the characterization of the first mirror that is being hit, continuing with mirrors of finite characterizations, simulating the finite DoE). The particle that starts in location $(0, y)$ first hits a particular mirror (the advice mirror) that throws it back to the plain to location (α, y), where $\frac{1}{4} < \alpha < \frac{1}{2}$ is a constant characterizing that mirror. The particle continues bouncing in the mirror system, simulating the Turing machine operation, where it starts the "computation" at the point (α, y) rather than $(0, y)$, and is confined to the $[\alpha, 1] \times [0, 1]$ part of the plane rather than to the unit square. When reaching the halting state of the Turing machine, the particle hits a mirror that throws it to a particular observable x coordinate, there all points are fixed. The output is defined as the y coordinate when reaching this observable x. Forcing the input and output to reside in observable areas, using for example Cantor set encoding, makes the halting state realizable. Another possible realization may be based on the recent optical realization of Baker's map [11].

Although it could have seemed that infinite precision was required to fully describe the associated computation, this is not the case because *linear precision suffices* for analog computation models [22]. That is, if one is interested in computing up to time q, both the mirror system and the location of the particle bouncing there are not required to be described (or measured) with more than q bits. Digital computers are still able to approximate the model with some round-off error. This property is in accordance with the sensitivity of chaotic systems to initial conditions (here, the mirror system), suggesting that the analog shift map is indeed a natural model of chaotic (idealized) physical dynamics.

Acknowledgements

I thank Allen Ponak from the University of Calgary, Jermey Schiff from Bar-Ilan University, and Shmuel Fishman from the Technion for helpful comments. Brian Hunt from the University of Maryland provided Fig. 1.

References

[1] V.I. Arnold and A. Avez, *Ergodic Problems of Classical Mechanics* (Benjamin, New York, 1968).

[2] G.L. Baker and J.P. Gollub, *Chaotic Dynamics: An introduction* (Cambridge University Press, Cambridge, 1990).

[3] J.L. Balcázar, J. Díaz and J. Gabarró, *Structural Complexity, Vols. I and II*, EATCS Monographs (Springer, Berlin, 1988–1990).

[4] J.L. Balcázar, R. Gavaldá, H.T. Siegelmann and E.D. Sontag, Some structural complexity aspects of neural computation, in: *IEEE Structure in Complexity Theory Conf.*, San Diego, CA (May 1993) 253–265.

[5] 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. AMS* **21** (1989) 1–46.
[6] R.L. Devaney, Dynamics of simple maps, in: *Proc. Symp. in Applied Mathematics* (1989) 1–24.
[7] C. Grebogi, E. Ott and J.A. Yorke, Chaos, strange attractors, and fractal basin boundaries in nonlinear dynamics, *Science* **238** (1987) 632–637.
[8] J. Guckenheimer and P. Holmes, *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields* (Springer, New York, 1983).
[9] J. Hertz, A. Krogh and R. Palmer, *Introduction to the Theory of Neural Computation* (Addison-Wesley, Redwood City, 1991).
[10] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA, 1979).
[11] J.P. Keating, J.H. Hannay and A.M.O. Dealmeida, Optical realization of the baker transformation, *Nonlinearity* **7** (1994) 1327–1342.
[12] P. Koiran, M. Cosnard and M. Garzon, Computability with low-dimensional dynamical systems, *Theoret. Comput. Sci.* **132** (1994) 113–128.
[13] C. Moore, Unpredictability and undecidability in dynamical systems, *Phys. Rev. Lett.* **64** (1990) 2354–2357.
[14] C. Moore, Generalized shifts: unpredictability and undecidability in dynamical systems, *Nonlinearity* **4** (1991) 199–230.
[15] H.E. Nusse and J.A. Yorke, *Dynamics: Numerical Explorations* (Springer, New York, 1994).
[16] E. Ott, *Chaos in Dynamical Systems* (Cambridge University Press, Cambridge, 1993).
[17] R. Penrose, *The Emperor's New Mind* (Oxford University Press, Oxford, 1989).
[18] H.T. Siegelmann, On the computational power of probabilistic and faulty neural networks, in: S. Abitebul and E. Shamir, eds., *Automata, Languages and Programming, Lecture Notes in Computer Science*, Vol. 820 (Springer, Jerusalem, 1994) 23–33.
[19] H.T. Siegelmann, On NIL: the software constructor of neural networks, *Parallel Process. Lett.*, to appear; previous version appeared in *Proc. 12th AAAI Conf.*, Seatle (August 1994) 887–882.
[20] H.T. Siegelmann, Computation beyond the turning limit, *Science* **268** (5210) (1995) 545–548.
[21] H.T. Siegelmann and E.D. Sontag, Turing computability with neural nets, *Appl. Math. Lett.* **4** (6) (1991) 77–80.
[22] H.T. Siegelmann and E.D. Sontag, Analog computation via neural networks, *Theoret. Comput. Sci.* **131** (1994) 331–360.
[23] H.T. Siegelmann and E.D. Sontag, On computational power of neural networks, *J. Comput. System Sci.* **50** (1995) 132–150; previous version appeared in *Proc. 5th ACM Workshop on Computational Learning Theory*, Pittsburgh (July 1992) 440–449.