A Dynamical Systems Perspective on the Relationship between Symbolic and Non-symbolic Computation

Whitney Tabor
DRAFT April 12, 2009

Department of Psychology U-1020
406 Babbidge Drive
University of Connecticut
Storrs, CT 06269-1020
whitney.tabor@uconn.edu

Abstract: Let $f_i : X \to X$ for $i = 1, \ldots, K$ be functions with domains in the continuous space, X. Given a one-sided infinite string of integers, $\sigma = \sigma_1\sigma_2 \ldots$, drawn from the set $\{1, \ldots, K\}$, let a process, M, begin in a specified initial state in X and apply the functions corresponding to the integers of σ in order. If each integer of σ corresponds to a legal function application, then the string, σ is said to be part of the "language" of M. In this way, classical dynamical systems, which operate on continuous spaces, can be interpreted as generators (or recognizers) of infinite-sentence formal languages on finite symbol sets. Such *dynamical automata* (Tabor, 2000) are closely related to the *dynamical recognizers* of Pollack (1991) and Moore (1998) and to the method of *symbolic dynamics* used to study the topologies of chaotic systems (Devaney, 1989). This paper shows that even a simple class of dynamical automata with linear transition functions generates formal languages of a wide range of types, including infinite state languages and non-Turing-computable languages. This sheds light on the implications of connectionist networks, a neurally-inspired class of mathematical models, for cognitive science. It also suggests exploring the deployment of formal language types in the parameter spaces of these models in order to gain high-level structural insight into bottom-up processes, like gradient descent learning, that operate on these parameter spaces.

Introduction

In the 1980s and 1990s, Jerry Fodor and Zenon Pylyshyn, on the one hand, and Paul Smolensky, on the other, had a debate about the relationship between symbolic and connectionist approaches to cognition. Part of the discussion centered on a notion of "compatibility". Fodor & Pylyshyn (1988, 1991) argued that the symbolic approach is right about the nature of cognition, and thus that, if connectionism is incompatible with the symbolic approach, it must be rejected. In fact, they argued that there is only one sense in which the connectionist approach might be compatible with the symbolic approach and that is as a "mere implementation": it might, for example, describe how the primitive symbols of symbol systems are instantiated in physical brains. Crucially, on this "mere im-

plementation" view, there is a clean division between the "lower" implementation level and the "higher" symbolic level of description such that all the causal relations at the symbolic level can be described without reference to properties at the implementation level. This claim has important implications for cognitive science: it suggests that cognitive scientists concerned with "high level" phenomena (presumably language, memory, conceptual structure, etc. fall under this heading) need to pay no attention to connectionism or any other implementation mechanism in order to successfully construct a theory of the phenomena. It also implies that high level cognition falls within the domain of "effective computation" as identified by the Church-Turing thesis: the theory of high level cognition can be fully formulated within the framework of so-called "Turing computation".

Smolensky (1988, 1995) took a contrary position, arguing for "incompatibility" between connectionism and the symbolic approach. He distinguished between "implementation" and "refinement", arguing that there is a way of doing connectionist modeling of cognition in which the models are not implementations but refinements of the symbolic models. He argues that the sense of implementation that Fodor & Pylyshyn must mean (in order to push the point about the irrelevance of connectionism to high level cognition) is the sense used in computer science, where a low-level implementation (e.g., an assembly language description) is an implementation of a high-level description (e.g., in BASIC or JAVA). A low-level refinement, by contrast, contains additional algorithmic information that is lacking in a high-level description. He says, "Far from the conclusion that '*nothing* can be gained by going to the lower-level account', there is *plenty* to be gained: completeness, precision, and algorithmic accounts of processing, none of which is generally available at the high level." (Smolensky, 1995, p. 168)

In a recent continuation of the debate, beim Graben (2004) argues that we should take advantage of insights from dynamical systems theory, especially the method of *symbolic dynamics*, to clarify the discussion. In particular, it is helpful to note that the state space of a connectionist model is generally a vector space so there is a continuum of possible states. Focusing on discrete update (iterated map) dynamics, the dynamical systems analysis method called *symbolic dynamics* adopts a finite partition of the state space and treats the partition indices as symbols. Thus, the iterating dynamical system on the vector space is associated with an iterating dynamical system on the symbol space. This correspondence gives rise to two ways of describing the system which beim Graben and Atmenspracher (2004) refer to as the "ontic" (vector space) and "epistemic" (symbolic alphabet) levels. beim Graben (2004) suggests that the ontic/epistemic distinction provides a good model for the subsymbolic/symbolic (low/high) distinction discussed in cognitive science. He goes on to suggest that the dynamical notion of *topological equivalence* (or *topological conjugacy*) can help formalize the concept of compatibility between descriptions. Two dynamical systems, $f : X \to X$ and $g : Y \to Y$ are *topologically conjugate* if there is a homeomorphism $h : X \to Y$ satisfying $g \circ h = h \circ f$. A *homeomorphism* is a continuous, one-to-one function. Topological conjugacy is a kind of structural correspondence: the states of two conjugate systems are in complete, point-to-point correspondence, and the patterns of transitions between states also correspond perfectly across the systems. A particularly strong kind of topological conjugacy occurs when the partition is a *generating partition*. A partition is generating if the future of the boundaries of the partition subdivides the space arbitrarily finely. In this case, almost all the information about the details of the continuous dynam-

ics can be reconstructed from the information about how the symbols are sequenced. For many dynamical systems, however, there is no generating partition and it is not possible to choose a single partition for which the symbolic dynamics reveals (almost) all the detail about the subsymbolic (continuous) dynamics. Such cases, beim Graben maintains, should count as cases of incompatibility between symbolic and subsymbolic dynamics. The fact that they exist among connectionist networks reveals, against Fodor & Pylyshyn's position, a fundamental incompatibility between the symbolic and connectionist approaches.

In this paper, I side with Smolensky and beim Graben in arguing for incompatibility between connectionist and symbolic systems. I also endorse beim Graben's emphasis on a dynamical systems perspective. However, I'll suggest that beim Graben's formalization of the notion of incompatibility is not the most useful one, and if we adopt a formulation more suitable to the central questions facing cognitive science, then the incompatibility is deeper than has been demonstrated in the aforementioned papers. In particular, the lack of completeness and precision that Smolensky mentions is certainly valid, but it does not seem to argue for a strong change in our approach to cognitive science. After all, the high-level algorithms that run on digital computers do not completely and precisely describe the physical actions of the computer chips; yet this lack does not, in any significant sense except for the very remote possibility of hardware errors, require that we pay attention to chip physics when describing the information processing behavior of digital computers. Beim Graben's examples of dynamical systems with incompatible epistemic and ontic dynamics are generally cases in which one partition induces symbolic dynamics corresponding to some familiar or simple algorithm (parsing a sentence, categorizing objects based on features, a simple finite-state process). None of the examples have generating partitions (at least under the parameterizations considered), so none exhibit strong informational equivalence across the levels. I will argue, however, that such cases should not interest us very much. They are all situations in which a vector space dynamical system can be used to generate a familiar algorithmic process via symbolic dynamics and there is nothing unexpected in the behavior at the symbolic level. In particular, Fodor & Pylyshyn's claim of separability seems to hold: if our interest is in the higher level description (i.e. the symbolic dynamics), then we don't need the vector space description to characterize these dynamics.

Smolensky (1995) also states that "algorithmic accounts of processing" are not available at the higher level for the connectionist systems he has in mind (p. 168). This *is* news if it means that no algorithm at all will suffice to describe the high-level behaviors of some of the systems. However, Smolensky does not provide evidence that such cases exist. A main purpose of this paper is to show that such cases do exist, and to suggest that they have implications for the type of phenomena we expect to observe in high level cognition. In making this point, I'll note that beim Graben's focus on the difference between systems with generating partitions and those which lack them is very helpful. However, somewhat confusingly, I'll argue that it is actually the cases *with* generating partitions that exhibit the kind of incompatibility we should be interested in. I reach this kind of conclusion by a simple argument: the case of greatest interest is the case in which a connectionist model does something that a classical symbol system cannot do; under certain conditions, connectionist models with real-valued weights compute non-Turing computable functions (Siegelmann, 1999; Turing, 1939). In fact, they can only do this when there *is* a generating partition. For this reason, I argue that "incompatibility" of classical and symbolic computation should be

associated with the availability, not the lack, of a generating partition.

The question of why students of cognition should care about the super-Turing behaviors is a critical one. Since the super-Turing behaviors depend on computations over reals that are not integers, infinite precision is required to realize them exactly; it is always possible to approximate them arbitrarily well with Turing computations. One may therefore think that, as far as empirical measurement goes, the difference between super-Turing behavior and classical Turing machine behavior will never be detectable. I'll argue that such a dismissal is unwise: despite the approximation possibility, we may be able to tell the difference between systems that are dominated by super-Turing behavior and those that are essentially equivalent to Turing machines. Moreover, there are relationships among computational systems that are invisible when one operates purely within the Turing computation world and that these relationships have implications for behavior. These observations make it preferable to conduct cognitive science in the super Turing regime. I'll return to these points in the Conclusion.

Relevant formal language classes

This section describes the classes of formal languages that are important in the discussion below.

A formal language is a set of finite-length strings drawn from a finite alphabet. The Chomsky hierarchy (Chomsky, 1956) classifies formal languages on the basis of the type of computing mechanism required to predict the allowable successor symbols following any given legal initial sequence. For example, the language L_1 consisting of the strings $\{ab, abab, ababab, \ldots\}$, can be fully predicted (or "generated") by a computing device with a finite number of distinct states. Such a device is called a "Finite State Automaton" (FSA). A more powerful type of device, the "Pushdown Automaton" (PDA) consists of a finite state automaton combined with an unbounded stack (first-in, last-out) memory. The top (last-in) symbol of the stack and the current state of the FSA jointly generate/predict the next symbol at each point in time. The language $L_2 = \{ab, aabb, aaabbb, \ldots\}$ (also called "$a^n b^n$") consisting of the strings with a positive whole number of "a"'s followed by the same number of "b"s, can be processed by a Pushdown Automaton, but not by a Finite State Automaton. Arguments have several times been advanced on linguistic grounds that human language processing employs something similar in computational capability to a PDA (Gazdar, 1981), though the current consensus is that a slightly more powerful device called a Tree Adjoining Grammar (Joshi & Schabes, 1996) provides the best characterization of the formal patterning of the syntax of natural languages (Savitch, 1987). An even more powerful device than those so far mentioned is the "Turing Machine" (TM). A TM consists of a FSA that controls an infinite tape memory (unbounded number of slots, the FSA controller can move step-by-step along the tape, either backwards or forwards, reading symbols and then either leaving them untouched or overwriting them as it goes). Any process that can be accomplished by manipulating a finite number of symbols through a finite number of states can be accomplished by a Turing Machine. In this sense, Turing Machines appear to correspond to the maximal case where so-called "effective computation" is concerned. Chomsky noted that the devices on this hierarchy define successively more inclusive sets of formal languages: every FSA language can be generated by a PDA; every PDA language

can be generated by a TM; but the reverse implications do not hold.[1]

Although they are not conventionally treated as a level on the Chomsky Hierarchy, there is a proper subset of the set of FSA languages that will be of relevance later in the present discussion: the Finite languages. These can be specified as a finite list of finite-length strings. There are also sets of strings that are not generated by any Turing Machine, the most powerful device on the Chomsky Hierarchy. These string-sets are sometimes called "super-Turing" languages (Siegelmann, 1999). These will also be relevant in the discussion below.

Formal Paradigm: Affine Dynamical Automata

Pollack (1991) defined *dynamical recognizers*: suppose $f_i : X \rightarrow X$ for $i = 1, \ldots, K$ are functions on the continuous space, X. Given a string of integers, σ, drawn from the set $\{1, \ldots, K\}$, we start the recognizer in a specified initial state in X and apply the functions corresponding to the integers of σ in order. If, at the end of applying the functions, the system is in a specified subset of the space (sometimes called the "accepting region"), then the string σ is said to be accepted by the dynamical recognizer. The set of strings accepted by the recognizer is a formal language. Moore (1998) explores conditions under which dynamical recognizers produce languages in various computational classes related to the Chomsky Hierarchy. One notable result is that, by choosing functions and regions judiciously, one can make dynamical recognizers for all Chomsky Hierarchy classes as well as for all super-Turing languages (Moore, 1998).

Here, I focus on a class of devices called "affine dynamical automata" (Tabor, 2000).
Def. An *affine dynamical automaton* (or ADA) is a device,

$$M_{\vec{h}_0} = (H, F, \vec{h}_0) \tag{1}$$

satisfying $H = \{\vec{h} \in \mathbf{R^N} : 0 \leq h_j \leq 1, j \in \{1, \ldots, N\}\}$, $F = \{f_1, f_2, \ldots, f_K\}$ is a set of affine functions that map H into itself, and $\vec{h}_0 \in H$ is the initial state. The domain, d_i, of each function, f_i, is $\{h \in H : f_i(h) \in H\}$. For a string, σ, of integers drawn from $\{1, \ldots, K\}$, the system starts at h_0 and, if possible, invokes the functions corresponding to the integers of σ in order. For some strings, there will come a point when the state of the system is not in the domain of the next function to be invoked. The complement of this string set is the language of M.

Affine dynamical automata are closely related to connectionist networks. Most connectionist networks employ a sigmoidal activation function. Iterated map networks related to the Simple Recurrent Network (Elman, 1990, 1991) can process infinite-state formal languages by using the contraction and expansion of the so-called "linear" (or central) section of the sigmoid to traverse fractal sets (Rodriguez, 1995; Rodriguez, Wiles, & Elman, 1999; Rodriguez, 2001; Tabor, 2000, 2003; Wiles & Elman, 1995). The affine maps of ADAs can be put to similar use. Thus, the study of ADAs may be informative about how connectionist networks can process complex languages.

[1] Chomsky (1956) also identified the set of Linear Bounded Automata which define a class of languages ("Context Sensitive Languages") that have the PDA languages as a proper subset, and are themselves a proper subset of the TM languages. Context Sensitive Languages are not of central concern in the present study.

A dynamical automaton is called "proper" if $H = \bigcup_{i=1}^{K} d_i$, i.e., if at least one of the automaton's functions can be applied at every point in H. Each proper dynamical automaton is associated with a "language" of one-sided infinite strings:
Def. Let $M = (H, F, \vec{h}_0)$ be a dynamical automaton. For $\sigma = \sigma_1\sigma_2\sigma_3 \ldots \in \Sigma^{\infty}$, let $\Phi[J]_\sigma = f_{\sigma_J}(f_{\sigma_{J-1}}(\ldots(f_{\sigma_2}(f_{\sigma_1}(h_0)))\ldots))$, where $\Phi[0]_\sigma = h_0$. If $\Phi[J]_\sigma$ exists for $J = 0, 1, 2, \ldots$, then σ is called an *infinite sentence* of M. The set of all infinite sentences of M is called *the infinite sentence language of* M.

Standard formal language theory (e.g., Hopcroft & Ullman, 1979), generally focuses on finite sentence languages. These are also generated by dynamical automata:
Def. Let $M = (H, F, \vec{h}_0)$ be a dynamical automaton. For $\sigma = \sigma_1\sigma_2\sigma_3 \ldots \in \{1, \ldots, K\}^*$, let $\Phi[J]_\sigma(h_0) = f_{\sigma_J}(f_{\sigma_{J-1}}(\ldots(f_{\sigma_2}(f_{\sigma_1}(h_0)))\ldots))$, where $\Phi[0]_\sigma(h_0) = h_0$. Suppose $\Phi[J]_\sigma(h_0)$ exists for $J = 0, 1, 2, \ldots |\sigma|$. Let $x = \Phi[|\sigma|]_\sigma(h_0)$. If $x \notin \bigcup_{i=1}^{K} d_i$, then σ is called a *finite sentence* of M. The set of all finite sentences of M is called *the finite sentence language of* M.
Def. The union of the infinite sentence language of M and the finite sentence language of M is called the *language of* M and is denoted $L(M)$.

Although standard formal language theory focuses on finite sentence languages, the theory extends naturally to infinite sentence languages and mixed (finite and infinite) sentence languages. For simplicity I focus on the largest class here.

Even some very simple classes of dynamical automata generate a rich variety of formal languages. The remainder of this paper focuses on two-function, one-dimensional linear dynamical automata.[2] Section 2 illustrates the range of computational behaviors exhibited by this very simple class of systems. Section 3 provides a parameter-space map, revealing the deployment of computational types within one portion of the parameter space of this dynamical class. The Conclusion returns to the matter of compatibility between symbolic and subsymbolic systems.

Range of phenomena within the class

Two function, one-dimensional affine dynamical automata ($f_1 = a_1h + b_1, f_2 = a_2h + b_2$) on $H = [0, 1]$ generate some members of each of the following classes: finite languages, finite-state languages, context free languages, and super-Turing languages. In fact, even two-function, one-dimensional *linear* dynamical automata ($f_1 = a_1h, f_2 = a_2h$) include all four of these types. This section gives an example of an affine automaton and a linear automaton of each type.

Finite Languages

A cobweb diagram illustrating one trajectory of the affine system

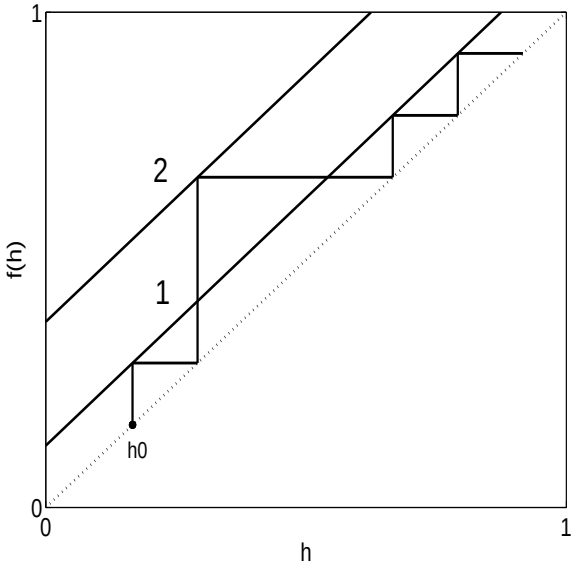
$$f_1(h) = h + \frac{1}{8}f_2(h) = h + \frac{3}{8}h_0 = 1/6 \tag{2}$$

is shown in Figure 1. This trajectory corresponds to the sentence, "1 2 1 1". Only sentences satisfying

$$\frac{1}{6} + \frac{3n}{8} + \frac{m}{8} > \frac{7}{8} \tag{3}$$

[2](Tabor, 2009) considers the more general case of two-function, one-dimensional affine dynamical automata.

Figure 1. A sample trajectory of $f_1(h) = h + 1/8, f_2(h) = h + 3/8, h_0 = 1/6$. The state space is the interval [0, 1]. The line segment labeled "i" is a plot of $f_i(h)$. The zig-zag line indicates the sample trajectory.



where $m \geq 0$ is the number of 1's and $n \geq 0$ is the number of 2s can be generated under this system. Since there are only a finite number of such cases, this automaton generates a finite language.

The linear system

$$\begin{aligned} f_1(h) &= \tfrac{4h}{3} \\ f_2(h) &= 2h \\ h_0 &= \tfrac{1}{4} \end{aligned} \tag{4}$$

generates the finite language {"111", "12", "211", "22"}.

Every linear system with $a_1 > 1$ and $a_2 > 1$ generates a finite language.

Finite State Languages

If the initial state in the previous example is replaced with $h_0 = 0$, then the system generates a finite-state language that is not a finite language. Figure 2 depicts a finite state device that generates/recognizes this language. I call this FSA BA for "Bernoulli Automaton" because one can think of it as a nonprobabilistic version of the Bernoulli Process. The circle corresponds to the single state of the machine. The system always starts in the state labelled "S". It moves between states by following arcs. The label on each arc indicates the symbol that is generated when the system traverses the arc. The Bernoulli Automaton generates all infinite sentences on the two-symbol alphabet $\Sigma = \{1, 2\}$. In fact, every linear system with $h_0 = 0$ generates $L(BA)$ and every linear system with $a_1 \leq 1$ and $a_2 \leq 1$ generates $L(BA)$ for all initial states.

Figure 2. The Bernoulli Automaton.

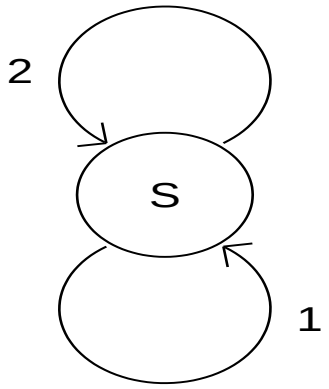


Figure 3a shows trajectories of the affine system

$$\begin{aligned} f_1(h) &= 2h \\ f_2(h) &= -\tfrac{1}{2}h + \tfrac{7}{6} \\ h_0 &= \tfrac{1}{5} \end{aligned} \tag{5}$$

This system is generated by the non-deterministic finite-state machine shown in Figure 3b.

Context Free Languages

Context free languages arise when contraction and expansion are precisely matched (Tabor, 2000). In particular, the following theorem states conditions under which linear dynamical automata generate context free languages:

> **Thm. 1** Let $DA = ([0,1], \{f_1(h) = a_1 h, f_2(h) = a_2 h\}, h_0 > 0)$ be a linear dynamical automaton. If
>
> $$a_1 = a_2^{-\frac{\alpha}{\beta}} \tag{6}$$
>
> where α and β are positive integers and a_1 and a_2 are not both 1, then $L(DA)$ is a context free language that is not a finite state language.

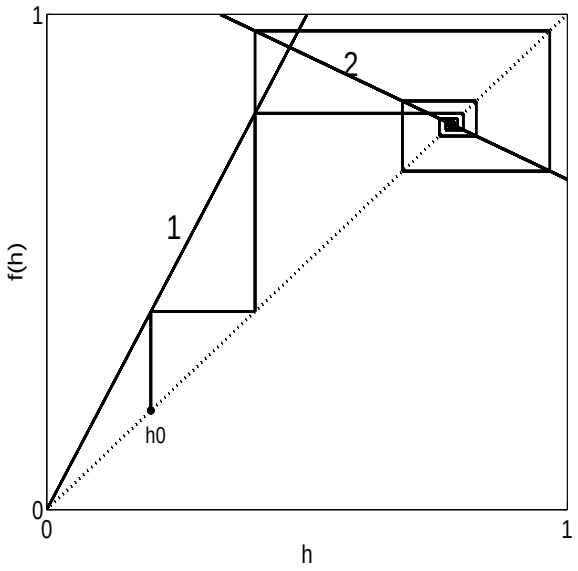
Appendix A1 provides a proof.

Figure 4 illustrates a trajectory and provides a context free grammar for a linear automaton that generates context free languages. A corresponding affine case is:

$$\begin{aligned} f_1(h) &= 2h - \tfrac{1}{3} \\ f_2(h) &= \tfrac{1}{2}h + \tfrac{2}{9} \\ h_0 &= 1 \end{aligned} \tag{7}$$

Figure 3. Trajectories (a) and finite state diagram (b) for the system, $f_1(h) = 2h, f_2(h) = -1/2h + 7/6, h_0 = 1/5$.

a. Trajectories.



b. Finite state diagram.

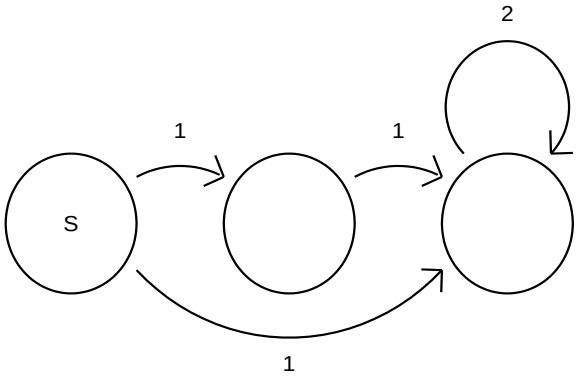
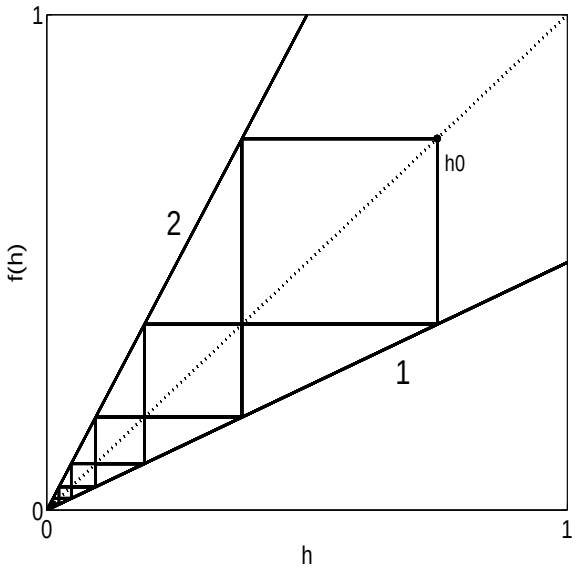


Figure 4. Trajectories (a) and context free grammar (b) for the system, $f_1(h) = 2h$, $f_2(h) = \frac{1}{2}h$, $h_0 = \frac{3}{4}$.

a. Trajectories.



b. Context Free Grammar.

S → 1 S 2
S → ϵ

Super-Turing Processes

Super Turing behavior occurs in the linear regime when the system has both expansion and contraction, but the two are not related by a rational power. To show this, some background is needed.

Consider a linear dynamical automaton, $DA = (H = [0,1], \{f_1(h) = a_1 h, f_2(h) = a_2 h\}, h_0 > 0)$ with $a_1 > 1$ and $0 < a_2 < 1$. Note that d_1, the domain of f_1 is $[0, 1/a_1]$. Thus, $b = 1/a_1$ is the boundary between the subset of H on which f_1 can apply and the subset of H on which it cannot. $1/a_1$ is called an *internal partition boundary* of the state space. Consider DA^{-1}, the inverse of DA:

$$DA^{-1} = ([0,1], \{f_1 = \frac{h}{a_1}, f_2 = \frac{h}{a_2}\}, \Sigma = [1,2], h_0 > 0) \tag{8}$$

DA^{-1} computes the inverse of the history of DA.

Def. The *future* of a state, $h \in H$, is the set $\{x \in H : x = \Phi_\sigma(h)$ for some $\sigma \in \{1, \ldots, K\}^*\}$.

That is, the future of a state h is the set of all states that the system can visit when started at h, including h itself.

> **Lemma** Let DA be an invertible dynamical automaton with inverse DA^{-1}. Consider an internal partition boundary, b of DA. If some subset of DA_b^{-1} is dense in an uncountable set in H, then DA generates uncountably many super Turing languages.

A proof of the Lemma is provided in Appendix A2. The Lemma provides a way of showing that some linear dynamical automata exhibit super Turing behavior.

> **Thm. 2** Let $DA = (H = [0,1], \{f_1(h) = a_1 h, f_2(h) = a_2 h\})$ be a linear dynamical automaton. If
>
> $$a_1 = a_2^{-\gamma} \tag{9}$$
>
> where γ is a positive irrational number, then there are states $h \in H$ for which $L(DA_h)$ is not generable/recognizable by any Turing device.

Theorem 2 is proved in Appendix A2.

Figure 5 shows a linear automaton that generates super-Turing languages. The illustration shows a single trajectory starting from the initial state, $h_0 = \frac{3}{4}$. I do not know if this particular initial state generates a super-Turing language. But there is bound to be a state very close to this initial state that generates a noncomputable language. Thus, the figure is illustrating an approximation of a super-Turing case. Even this approximation appears to be less regular than the trajectories in the finite, finite-state, and context-free examples discussed above.

It is a little easier to show the existence of super Turing behaviors in the affine case. The chaotic Baker Map (Devaney, 1989) is equivalent to the affine automaton,

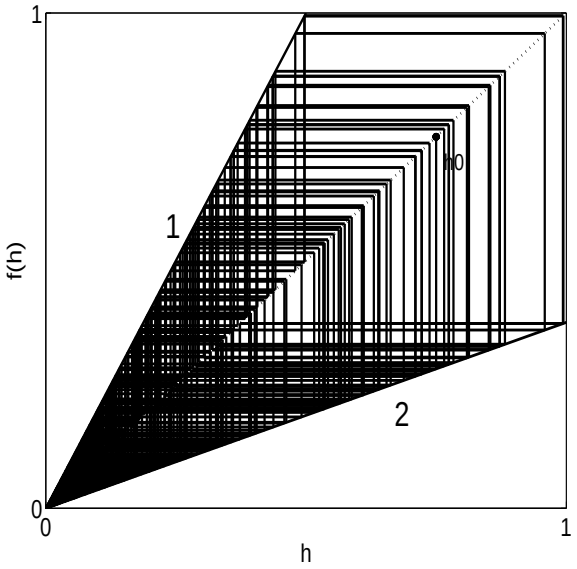
$$\begin{array}{rcl} f_1(h) & = & 2h \\ f_2(h) & = & 2h-1 \end{array} \tag{10}$$

The inverse of the Baker Map (BM^{-1}) is

$$\begin{array}{rcl} f_1(h) & = & \frac{1}{2}h \\ f_2(h) & = & \frac{1}{2}h + \frac{1}{2} \end{array} \tag{11}$$

The Baker Map has one internal partition boundary at h = 1/2. The union of the zero'th through the n'th iterates of 1/2 under (BM^{-1}) consists of the points $\{k/2^n\}$ for $k = 1 \ldots 2^{n-1}$. Therefore any point in H can be arbitrarily well approximated by a finite iterate of BM^{-1} and the future of 1/2 is dense in H. Thus there are uncountably many initial states h, for which $L(BM_h)$ is not describable by a Turing Machine. A similar argument applies to the Tent Map (Devaney, 1989).

Figure 5. A single trajectory of the system $f_1(h) = 2h$, $f_2(h) = \frac{1}{2^{\sqrt{2}}}h$, $h_0 = \frac{3}{4}$.



Parameter Space Map

The results of the preceding section justify the construction of the parameter space map shown in Figure 6. The slope of the first function is encoded along the horizontal dimension and the slope of the second is encoded along the vertical.[3] The map indicates the type of language associated with most initial states under the parameter setting. The blocks with shaded bands contain settings that generate context-free languages and settings that generate super-Turing languages. When the exponent, γ, is rational, then all initial states except $h_0 = 0$ are associated with context free languages that are not finite state languages. The greyscale indicates, for rational values of γ, the number of symbols required to write a grammar of the language, with lighter shading corresponding to languages requiring fewer symbols. The super-Turing languages occur when γ is irrational, but only for a proper subset (an uncountably infinite proper subset) of the initial states (see discussion of the Baker Map in the Conclusion).

Conclusion

The paper has provided evidence that a rich variety of computational structures occur in even very simple dynamical computing devices. All two-function linear dynamical automata were considered, supporting the presentation of a parameter space map for this subclass.

A companion paper extends the theory to all two-function affine cases (Tabor, 2009).

The Introduction argued that the possibility of super-Turing computation in dynamical automata (and hence in related connectionist devices) implies that there is an incom-

[3]Maps with one or both slopes negative generate only finite or finite state languages and are not shown.

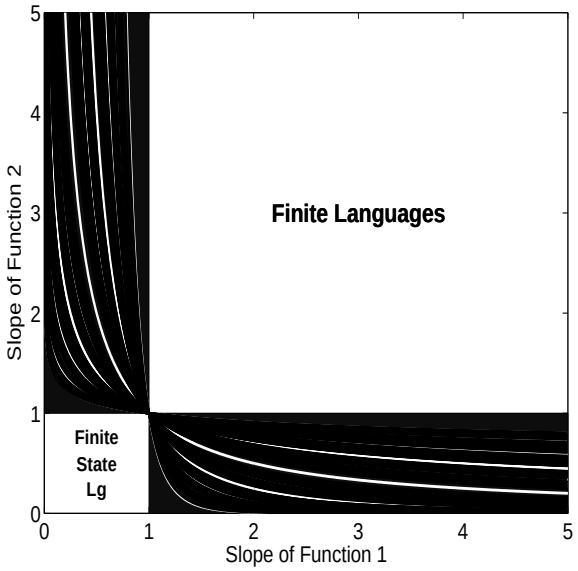


Figure 6. Deployment of language types in the parameter space of two-function linear dynamical automata.

patibility between the connectionist and classical symbolic paradigms for cognitive science. I suggested in the introduction that this form of incompatibility is a kind that cognitive science should take particular note of.

The analysis presented in the body of this paper makes it evident that super-Turing behaviors are very prevalent: they are uncountably numerous, while Turing behaviors are countable. However, the super-Turing behaviors are subtle and hard to detect. For example, even in an automaton with super Turing initial states dense in the whole state space, it is difficult to know if a particular initial state actually generates a super-Turing process. Even some processes with very complex trajectories can be generated by Turing devices. For example, the Baker Map mentioned above has the property that it generates, in succession, the digits of the base 2 representation of its initial state. If it is started at an irrational initial state like $1/\sqrt{2}$, it will travel on a complex, nonrepeating path. Nevertheless, there are algorithms that compute the digits of $\sqrt{2}$ and hence of $1/\sqrt{2}$. An interesting question is whether there are trajectories that are chaotic (e.g., in the sense of Devaney, 1989) that are nevertheless Turing computable. The non-Turing-computable nature of the super-Turing processes lies, evidently, in the fact that they contain endless new information in their initial states.

I raised the question in the Introduction whether cognitive science should care about the super-Turing processes, especially given their fleeting nature. For two reasons, I think it should:

First, although they do not occur in the linear realm, there may be cases among more complex (e.g. affine) automata in which a volume of parameter space of nonzero measure contains no Turing devices at all. Even a noisy system in such a region would be dominated by the super-Turing dynamics. Perhaps it would be possible to show, in such a case, that there exists no single Turing device that characterizes the systems's movement for a period

of any great duration. The existence of such regimes in the model would suggest that we test for their existence in people.

Second, the perspective which includes the super-Turing devices has an advantage over the perspective that ignores them because it places the Turing computation systems in a metric space in which real-valued distances between devices are defined. It appears that the parameter spaces map continuously to behavior space so that small changes in the parameters give rise to small changes in the behaviors. Such a situation is desirable for explaining how a system without structure can metamorphose gracefully into one that has structure (as may happen in human development) or how a system with one kind of structure can metamorphose gracefully into a system with a different kind of structure (as may happen in conceptual reorganization and language change). Some headway has been made in studying the emergence of structure via the bottom-up learning algorithms employed by connectionist networks (usually gradient descent methods). The present perspective suggests a way of studying the progress of such algorithms with respect to high level structural properties of their behavior by watching them traverse annotated versions of their parameter spaces like Figure 6—see (Tabor, 2003). Theories couched purely in the Turing computation framework seem hard pressed to make useful claims about the proximity of one Turing computation system to another and thus fail to offer this kind of perspective on the emergence of structure.

On the whole, then, the perspective adopted here indicates a surprisingly intimate relationship between symbolic and connectionist computation. Provided we accept the view that these simple first-order function maps tell us something about connectionsim in the general case, where nonlinear maps are more common, the results indicate that connectionism cannot be viewed either as an implementation or a subset of symbolic computation. In this sense, the two are incompatible. On the other hand, as the current results emphasize, symbolic computation (computation over the integers) can be viewed as a proper subset of dynamical computation (computation over the reals). The results suggest that it may be a particularly important subset. Among the classes of computing devices in focus here (finite, finite-state, context-free, and super-Turing), the context-free devices are the ones that have the strongest claim to cognitive relevance and universal insightfulness. Pushdown stack operations are plausible centrally involved in natural language processing. They also form the core of most digital computer programming. They also appear to govern the patterns in plant growth (Lindenmayer & Prusinkiewicz, 1989). Within the space of affine automata, context free behavior is associated with symmetries in parameter settings: if the functions perfectly invert each other's operations, then the simplest context free language ($a^n b^n$) occurs. The dynamical perspective thus suggests a reason why it seems, as Fodor and Pylyshyn have it, that symbolic computation is the only game in town: it's an effective game; it's an ideal game; life systems appear to cleave to it. But the present considerations suggest that it would be unwise to limit our attention to it, on this account. Instead, we will achieve more insight if we adopt a broader view and seek to derive the instances of Turing Machine computation (or approximately Turing Machine computation) within this broader view.

Appendix A1

Thm. 1 Let $DA = ([0,1], \{f_1(h) = a_1 h, f_2(h) = a_2 h\}, h_0 > 0)$ be a linear dynamical automaton. If

$$a_1 = a_2^{-\frac{\alpha}{\beta}} \tag{12}$$

for α, β positive integers and $a_1, a_2 \neq 1$. Then $L(DA)$ is a context free language that is not a finite state language.

Proof: Without loss of generality, we can assume $a_1 > 1$ and $a_2 < 1$. Choose two real numbers, m and n satisfying

$$m\beta + n\alpha = 1 \tag{13}$$

For example, we could take $m = 0$ and $n = \frac{1}{\alpha}$. Let

$$u = a_1^n a_2^m \tag{14}$$

Therefore

$$\begin{aligned} a_1 &= a_1^{m\beta - n\alpha} \\ &= a_1^{-n\alpha} a^{m\beta} \\ &= a_1^{-n\alpha} (a^{-\beta})^{-m} \\ &= (a_1^n a_2^m)^{-\alpha} \\ &= u^{-\alpha} \end{aligned} \tag{15}$$

Similarly

$$a_2 = u^{\beta} \tag{16}$$

Note that $u = a_1^{-\alpha}$ where α is a positive integer and $a_1 > 1$, so $u < 1$.

Let PDA be a pushdown automaton with a one-symbol stack alphabet. Whenever PDA reads a 2, it pushes β symbols onto its stack. Whenever PDA reads a 1, it pops α symbols off its stack. For $x \in [0,1]$, let

$$q(x) = \lceil log_u(x) \rceil \tag{17}$$

where $\lceil y \rceil$ denotes the least integer greater than or equal to y. At the start of processing, the stack of PDA has $q(h_0)$ symbols on it. For $\sigma \in \Sigma^{\infty}$, if PDA makes a possible move at every symbol of σ in sequence (i.e., it never encounters a 1 with fewer than α symbols on its stack), then PDA is said to recognize σ. Let $L(PDA)$ be the set of strings recognized by PDA.

Let $\sigma \in \Sigma^{\infty}$ be a sentence of $L(DA)$. Consider $\sigma[n]$ the length n prefix of σ for $n \in N$. Suppose that, upon processing the prefix $\sigma[n]$, PDA has j symbols on its stack and M is at a point $x \in [0,1]$ where $q(x) = j$. Then, if the next symbol is a 2, PDA will add β symbols to its stack. Similarly, the new state of M will be $x_2' = f_2(x) = a_2 x = u^{\beta} x$. Therefore, $q(x_2') = \lceil log_u u^{\beta} x \rceil = \beta + \lceil log_u(x) \rceil = j + \beta$. Likewise, if the next symbol is a 1, then PDA will remove α symbols from its stack. Similarly, the new state of M will be $x_1' = f_1(x) = a_1 x = u^{-\alpha} x$. Therefore, $q(x_1') = \lceil log_u u^{-\alpha} x \rceil = -\alpha + \lceil log_u(x) \rceil = j + \alpha$. Therefore, since $q(h_0)$ was the number of symbols on the stack at the start of processing, $q(x)$ equals the number of symbols on the stack at every step, provided that every legal move of M is a legal move of PDA and vice versa. In other words, it remains to be shown that PDA and DA always generate/accept the same next symbol. When the stack has α

or more symbols on it, both 1 and 2 are possible next inputs. But when the stack has fewer than α symbols on it, only 2 is a possible next input. Likewise, if $q(x) \geq \alpha$ then the state of M is in the domains of both f_1 and f_2, but when $q(x) < \alpha$ then

$$\begin{array}{rcl} \lceil log_u(x) \rceil & < & \alpha \\ log_u(x) & < & \alpha \\ x & > & u^{\alpha} \end{array} \tag{18}$$

since $u < 1$ and $\alpha > 0$. Thus

$$\begin{array}{rcl} x^{-1} & < & u^{-\alpha} \\ x^{-1} & < & a \\ x & > & a_1^{-1} \end{array} \tag{19}$$

Since a_1^{-1} is the upper bound of the domain of f_1, only f_2 can be applied. Thus $L(M) = L(PDA)$. By a well-known theorem (Hopcroft & Ullman, 1979), $L(PDA)$ is a context free language. Since PDA can have an unbounded number of symbols on its stack during the processing of legal strings from its language, $L(PDA)$ is not a finite state language. Thus $L(M)$ is a context free language that is not a finite state language. $\diamond$

Appendix A2

Lemma Let DA be an invertible dynamical automaton with inverse DA^{-1}. Consider an internal partition boundary, b of DA. If some subset of DA_b^{-1} is dense in an uncountable set in H, then DA generates uncountably many super Turing languages.

Proof: Consider the future, F_b, of the internal partition boundary, $b = 1/a_1$ under DA^{-1}. Suppose that some part of this future is dense in an uncountable set. Since the future itself is countable, there must be an uncountable number of points that are not part of the future, but which are pairwise separated from one another by points in F_b. Consider any pair of such points, x and y and the separating point, $B \in F_b$. Consider iterating DA, the forward map, simultaneously from the initial points x, y, and B. Since B lies strictly between x and y and the functions of DA are linear, if the same symbol is chosen for each of the three trajectories at every step, then each iterate of the future of B will always lie strictly between each iterate of x and y. For some σ, $\Phi_\sigma(B) = b$. Thus the futures of x and y eventually lie on opposite sides of the internal partition boundary. Therefore the futures of x and y are nonidentical. Consequently $L(DA_x) \neq L(DA_y)$. Since this is true of uncountably many pairs of points, there must be an uncountable number of distinct languages generated by DA. Since the Turing processes are countable, uncountably many of these languages must be Super Turing languages. $\diamond$

Thm. 2 Let $DA = (H = [0,1],\ \{f_1(h) = a_1 h, f_2(h) = a_2 h\})$ be a linear dynamical automaton. If

$$a_1 = a_2^{-\gamma} \tag{20}$$

where γ is a positive irrational number, then there are states $h \in H$ for which $L(DA_h)$ is not generable/recognizable by any Turing device.

Proof: Without loss of generality, assume $a_1 > 1$. By the Lemma, it will suffice to show that the future, F, of the point, $1/a_1$, under DA^{-1} is dense in an interval of positive length.

In fact, F is dense in H itself. Let $\alpha_1 = 1/a_1$ and $\alpha_2 = 1/a_2$. Note that $\alpha_1 = \alpha_2^{-\gamma}$. Consider $y = \alpha_1^j \alpha_2^k (1/a_1)$, for j and k nonnegative integers and $k \leq \gamma(j+1)$, a point in the future of $1/a_1$ under DA^{-1}. I will show that for every $x \in H$ and every ϵ, there exists y satisfying the conditions just mentioned, with $|y - x| < \epsilon$. Note that

$$\begin{aligned} log_{\alpha_2} y &= log_{\alpha_2} \alpha_1^j \alpha_2^k (1/a_1) \\ &= log_{\alpha_2} (\alpha_2^{-\gamma})^j \alpha_2^k \alpha_1 \\ &= (log_{\alpha_2} \alpha_1) - \gamma j + k \end{aligned} \tag{21}$$

$f(z) = z - \gamma \ (mod\ 1)$ is an irrational rotation on [0, 1]. Therefore the set $\{z - \gamma j \ (mod\ 1) : j = 0, 1, 2, \ldots\}$ is dense in [0, 1]. Thus there is a nonnegative integer j such that $log_{\alpha_2}\alpha_1 - \gamma j \ (mod\ 1)$ is within ϵ of $log_{\alpha_2} x \ (mod\ 1)$. It follows that there is a nonnegative integer k ($k \leq \gamma(j+1)$) such that $(log_{\alpha_2}\alpha_1) - \gamma j + k$ is within ϵ of $log_{\alpha_2} x$. Since log_{α_2} is contractive on (0, 1), it also follows that $y = \alpha_1^j \alpha_2^k (1/a_1)$ is within ϵ of x. Thus F is dense in [0, 1]. $\diamond$

References

beim Graben, P. (2004). Incompatible implementations of physical symbol systems. *Mind and Matter*, *2*(2), 29–51.

beim Graben, P., & Atmenspracher, H. (2004). *Complementarity in classical dynamical systems.* (arXiv: nlin.CD/0407046)

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, *2*(3), 113-124. (A corrected version appears in Luce, Bush, and Galanter, eds., 1965 *Readings in Mathematical Psychology, Vol. 2*)

Devaney, R. L. (1989). *An introduction to chaotic dynamical systems, 2nd ed.* Redwood City, CA: Addison-Wesley.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*, 179-211.

Elman, J. L. (1991). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, *7*, 195–225.

Gazdar, G. (1981). On syntactic categories. *Philosophical Transactions (Series B) of the Royal Society*, *295*, 267-83.

Hopcroft, J. E., & Ullman, J. D. (1979). *Introduction to automata theory, languages, and computation.* Menlo Park, California: Addison-Wesley.

Joshi, A. K., & Schabes, Y. (1996). Tree-adjoining grammars. In G. Rozenberg & A. Salomaa (Eds.), *Handbook of formal languages* (Vol. 3, pp. 69–123). New York: Springer-Verlag.

Lindenmayer, A., & Prusinkiewicz, P. (1989). Developmental models of multicellular organisms: a computer graphics perspective. In C. G. Langton (Ed.), *Artificial life* (pp. 221–250). Redwood City, CA: Addison-Wesley. (A proceedings volume in the Santa Fe Institute Studies in the Sciences of Complexity)

Moore, C. (1998). Dynamical recognizers: Real-time language recognition by analog computers. *Theoretical Computer Science*, *201*, 99–136.

Pollack, J. B. (1991). The induction of dynamical recognizers. *Machine Learning*, *7*, 227-252.

Rodriguez, P. (1995). *Representing the structure of a simple context-free language in a recurrent neural network: A dynamical systems approach.* http://crl.ucsd.edu/newsletter.html. (On-line Newsletter of the Center for Research on Language, University of California, San Diego.)

Rodriguez, P. (2001). Simple recurrent networks learn context-free and context-sensitive languages by counting. *Neural Computation*, *13*(9).

Rodriguez, P., Wiles, J., & Elman, J. (1999). A recurrent neural network that learns to count. *Connection Science*, *11*(1), 5–40.

Savitch, W. J. (Ed.). (1987). *The formal complexity of natural language.* Norwell, MA: Kluwer.

Siegelmann, H. T. (1999). *Neural networks and analog computation: Beyond the turing limit.* Boston: Birkhäuser.

Smolensky, P. (1988). On the proper treatment of connectionism. *Behavioral and Brain Sciences*, *11*(1), 1–74.

Smolensky, P. (1995). Connectionism, constituency, and the language of thought. In C. MacDonald & G. MacDonald (Eds.), *Connectionism. debates on psychological explanation.* Oxford: Blackwell.

Tabor, W. (2000). Fractal encoding of context-free grammars in connectionist networks. *Expert Systems: The International Journal of Knowledge Engineering and Neural Networks*, *17*(1), 41-56.

Tabor, W. (2003). Learning exponential state growth languages by hill climbing. *IEEE Transactions on Neural Networks*, *14*(2), 444-446.

Tabor, W. (2009). *Affine dynamical automata.* (Ms., University of Connecticut Department of Psychology)

Turing, A. (1939). Systems of logic based on ordinals. *Procedings of the London Mathematical Society.*

Wiles, J., & Elman, J. (1995). Landscapes in recurrent networks. In J. D. Moore & J. F. Lehman (Eds.), *Proceedings of the 17th annual cognitive science conference.* Lawrence Erlbaum Associates.