Dynamical Recognizers: Real-time Language Recognition by Analog Computers

Cristopher Moore
Santa Fe Institute
moore@santafe.edu

November 3, 1999

Abstract

We consider a model of analog computation which can recognize various languages in real time. We encode an input word as a point in $\mathbb{R}^d$ by composing iterated maps, and then apply inequalities to the resulting point to test for membership in the language.

Each class of maps and inequalities, such as quadratic functions with rational coefficients, is capable of recognizing a particular class of languages. For instance, linear and quadratic maps can have both stack-like and queue-like memories. We use methods equivalent to the Vapnik-Chervonenkis dimension to separate some of our classes from each other: linear maps are less powerful than quadratic or piecewise-linear ones, polynomials are less powerful than elementary (trigonometric and exponential) maps, and deterministic polynomials of each degree are less powerful than their non-deterministic counterparts.

Comparing these dynamical classes with various discrete language classes helps illuminate how iterated maps can store and retrieve information in the continuum, the extent to which computation can be hidden in the encoding from symbol sequences into continuous spaces, and the relationship between analog and digital computation in general.

We relate this model to other models of analog computation; in particular, it can be seen as a real-time, constant-space, on-line version of Blum, Shub and Smale's real-valued machines.

1 Introduction

Suppose that for each symbol a in a finite alphabet, we have a map f_a acting on a continuous space. Given an input word, say $abca$, we start with an initial point and apply the maps f_a, f_b, f_c and f_a in that order. We then accept or reject the input word depending on whether or not the resulting point x_{abca} is in a particular subset of the space; the set of words we accept forms a *language* recognized by the system.

We will call such systems dynamical recognizers; they were formally defined by Jordan Pollack in [36]. To define them formally, we will use the following notations (slightly different from his):

A^* is the set of finite words in an alphabet A, with ϵ the empty word. If w is a word in A^*, then $|w|$ is its length and w_i is the ith symbol, $1 \leq i \leq |w|$. We write a^k for a repeated k times. The concatenation of two words $u \cdot v$, or simply uv, is $u_1 \cdots u_{|u|} v_1 \cdots v_{|v|}$.

Suppose we have a map f_a on $\mathbb{R}^d$ for each symbol $a \in A$. Then for any word w, $f_w = f_{w_{|w|}} \circ \cdots \circ f_{w_2} \circ f_{w_1}$ is the composition of all the f_{w_i}, and $x_w = f_w(x_0)$ is the encoding of w into the space where $x_0 = x_\epsilon$ is a given initial point.

Then a *real-time deterministic dynamical recognizer* ρ consists of a space $M = \mathbb{R}^d$, an alphabet A, a function f_a for each $a \in A$, an initial point x_0, and a subset $H_{\text{yes}} \subset M$ called the *accepting subset.* The *language recognized by* ρ is then $L_\rho = \{w \,|\, x_w \in H_{\text{yes}}\}$, the set of words for which iterating the maps f_{w_i} on the initial point yields a point in the accepting set H_{yes}.

For example, suppose $M = \mathbb{R}$, $A = \{a, b\}$, $f_a(x) = x + 1$, $f_b(x) = x - 1$, $x_0 = 0$, and $H_{\text{yes}} = [0, \infty)$. Then if $\#_a(w)$ and $\#_b(w)$ are the number of a's and b's in w respectively, $x_w = \#_a(w) - \#_b(w)$ and $L(\rho)$ is the set of words for which $\#_a(w) \geq \#_b(w)$.

We can also define *non-deterministic* dynamical recognizers: for each $a \in A$, let there be several choices of function $f_a^{(1)}, f_a^{(2)}$ etc. Then we accept the word w if there exists a set of choices that puts x_w in H_{yes}, i.e.

$$x_w^{(k)} = f_{w_{|w|}}^{(k_{|w|})} \circ \cdots \circ f_{w_2}^{(k_2)} \circ f_{w_1}^{(k_1)}(x_0) \in H_{\text{yes}} \text{ for some sequence } k.$$

In this paper, we will look at classes of dynamical recognizers and the corresponding language classes they recognize. For a given class $\mathcal{C}$ of functions and a given subset $U \subset \mathbb{R}$ such as $\mathbb{Z}$ or $\mathbb{Q}$, we define the class $\mathcal{C}(U)$ as the set of languages recognized by dynamical recognizers where:

1) $x_0 \in U$,

2) H_{yes} is defined by a Boolean function of a finite number of inequalities of the form $h(x) \geq 0$, and

3) the h and f_a for all a are in $\mathcal{C}$ with coefficients in U.

We will indicate a non-deterministic class with an $\mathbf{N}$ in front. In particular:

$\mathbf{Poly}_k(U)$ and $\mathbf{NPoly}_k(U)$ are the language classes recognized by deterministic and non-deterministic polynomial recognizers of degree k with coefficients in U.

$\mathbf{Lin}(U) = \mathbf{Poly}_1(U)$ and $\mathbf{NLin}(U) = \mathbf{NPoly}_1(U)$ are the deterministic and non-deterministic linear languages.

$\mathbf{Poly}(U) = \cup_k \mathbf{Poly}_k(U)$ and $\mathbf{NPoly}(U) = \cup_k \mathbf{NPoly}_k(U)$ are the deterministic and non-deterministic polynomial languages of any degree.

PieceLin(U) and **NPieceLin**(U) are the languages recognized by piecewise-linear recognizers with a finite number of components, whose coefficients and component boundaries are in U.

Elem(U) and **NElem**(U) are languages recognized by elementary functions, meaning compositions of algebraic, trigonometric, and exponential functions, whose constants can be written as elementary functions of numbers in U.

We will take U to be $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$. We will leave U out if it doesn't affect the statement of a theorem.

2 Memory, encodings, analog computation, and language

There are several reasons one might want to study such things.

First, by restricting ourselves to real time (i.e. one map is applied for each symbol, with no additional processing between) and only allowing measurement at the end of the input process, we are in essence studying *memory*. If a dynamical system is exposed to a series of influences over time (a control system, say, or the external environment), what can we learn about the history of those influences by performing measurements on the system afterwards? What kinds of long-time correlations can it have? What kinds of information storage and retrieval can it do? For instance, we will show that linear and quadratic maps can have both stack-like (last in, first out) and queue-like (first in, first out) memories.

Secondly, a number of recent papers [30, 32, 10, 1] have shown that various kinds of iterated maps (piecewise-linear, differentiable, C^∞, analytic, etc.) in low dimensions are capable of various kinds of computation, including simulation of universal Turing machines. However, in and of themselves, these statements are ill-defined; for a continuous dynamical system to simulate discrete computation, we need to define an interface between the two. We illustrate this conceptually in figure 1: we encode a discrete input w as a point $x = f(w)$ in the continuous space, iterate the continuous dynamics until some halt condition is reached, and then measure the result by mapping the continuous state back into a discrete output $h(x)$.

The problem is that with arbitrary encoding and measurement functions, the identity function, with no dynamics at all, can recognize any language! All we have to do is hide all the computation in the encoding itself: let $f(w) = 1$ if $w \in L$ and 0 otherwise, and let $h(x)$ be 'yes' if $x > 0$. We can do the same thing on the measurement side by letting $h(x_w)$ be 'yes' if $w \in L$ and 'no' otherwise.

Clearly there is something unreasonable about such encoding and measurement functions; the question is how to define reasonable ones. Most of these papers use the best-known encoding from discrete to continuous, namely the digit sequence $x = .a_0a_1\ldots$ of a real number. Finite words correspond to blocks

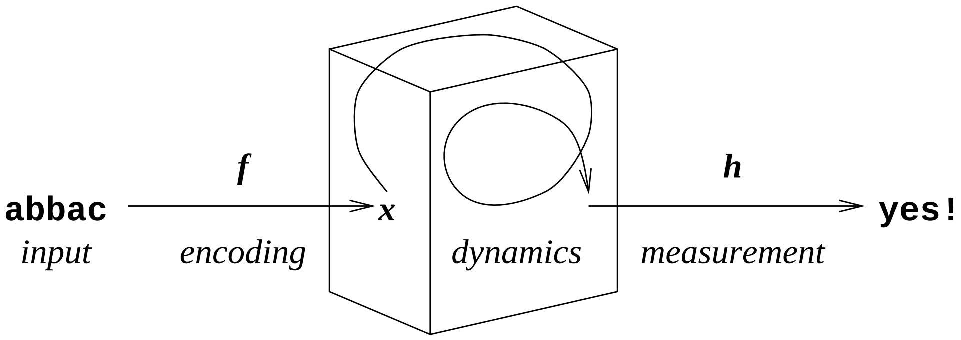

Figure 1: The interface between discrete and comtinuous computation: encoding a discrete word in a continuous space, evolving the dynamics, and performing a measurement to extract a discrete result.

in the unit interval. If we add gaps between the blocks, we get a Cantor set; for instance, the middle-thirds Cantor set consists of those reals with no 1's in their base-3 expansion.

This encoding can be carried out by iterating affine maps: if $A = \{0, 2\}$, let $x_0 = 0.1$, $f_0(x) = x/3$ and $f_2(x) = x/3 + 2/3$. Then $f_w(x_0) = .w_{|w|} \ldots w_2 w_1 1$ is the point in the center of the block corresponding to w. We could say then that this encoding is reasonable to whatever extent that affine maps are.

This suggests the following thesis: that reasonable encodings consist of reasonable maps, iterated in real time as the symbols of the word are input one by one. If we accept this, then this paper is about how much computation can be hidden in the encoding and measurement process, depending on what kinds of maps are allowed.

Thirdly, there is an increasing amount of interest in models of analog computation, such as Blum, Shub, and Smale's flowchart machines with polynomial maps and tests [3] and other models [28] with linear or trigonometric maps as their elementary operations. In this context, dynamical recognizers form a hierarchy of analog computers with varying sets of elementary operations. We show below that dynamical recognizers can be thought of as on-line BSS-machines with constant space.

Finally, recurrent neural networks are being studied as models of language recognition [36] for regular [16], context-free [13, 41], and context-sensitive [39] languages, as well as fragments of natural language [14], where grammars are represented dynamically rather than symbolically. The results herein then represent upper and lower limits on the grammatical capabilities of such networks in real time, with varying sorts of nonlinearities. Perhaps these are 'baby steps' toward understanding the cognitive processes of experience, imagination, and communication, so important to our everyday lives [33], in a dynamical, rather than digital, way.

3 Discrete computation classes

We will relate our dynamical classes to the following language classes from the standard theory of discrete computation [21, 34]:

Reg, the *regular languages*, are recognizable by finite-state automata (FSAs) and are representable by expressions using concatenation, union, and the Kleene star $*$ (iteration 0 or more times). For instance, $(a+ba)^*$ consists of those strings where two adjacent b's never appear and which end with an a.

CF, the *context-free languages*, are recognizable by *pushdown automata* (PDAs), which are FSAs with access to a single stack memory. A word is accepted either when the FSA reaches a certain state or when the stack is empty. Context-free languages are also generated by *context-free grammars* where single symbols are replaced by strings.

For instance, the Dyck language $\{\epsilon, (), (()), ()(), \ldots\}$ of properly matched parentheses is generated from an initial symbol X by a grammar where the initial symbol X can be replaced with $(X)X$ or erased. It is recognized by a PDA that pushes a symbol onto its stack when it reads a "(" and pops one when it reads a ")". Since this PDA is deterministic, this language is actually in **DCF**, the *deterministic context-free languages*.

CS, the *context-sensitive languages*, are recognizable by Turing machines which only use an amount of memory proportional to the input size. For instance, the language $\{x^p\}$ of words of prime length is context-sensitive.

We have $\mathbf{Reg} \subset \mathbf{DCF} \subset \mathbf{CF} \subset \mathbf{CS}$, with all containments proper.

$\mathbf{TIME}(f(n)), \mathbf{NTIME}(f(n)), \mathbf{SPACE}(f(n))$, and $\mathbf{NSPACE}(f(n))$ are the languages recognizable by a multi-tape Turing machine, deterministic or non-deterministic, using only time or memory proportional to $f(n)$ where n is the length of the input. For instance, $\mathbf{NSPACE}(n) = \mathbf{CS}$, and $\cup_k \mathbf{TIME}(n^k)$ and $\cup_k \mathbf{NTIME}(n^k)$ are the (distinct?) classes **P** and **NP** of problems that can be solved deterministically and non-deterministically in polynomial time — not to be confused with the **Poly** and **NPoly** of this paper!

$\mathbf{NC}_k$ is the class of languages recognizable by a Boolean circuit of depth $\log^k n$ and polynomial size, or equivalently by a parallel computer with a polynomial number of processors in time $\log^k n$. The union $\mathbf{NC} = \cup_k \mathbf{NC}_k$, Nick's Class, is the set of problems that can be solved in polylogarithmic parallel time; it is believed to be a proper subset of **P**.

4 Closure properties and general results

Closure properties are a useful tool in language theory. We say a class $\mathcal{C}$ of languages is closed under a given operator (union, intersection, complementation, and so on) if whenever languages L_1, L_2 are in $\mathcal{C}$ then $L_1 \cup L_2, L_1 \cap L_2, \overline{L_1} \ldots$ are also.

Then we can prove the following easy lemmas. Most of these are axiomatic in nature, and would be equally true for any recognition machine with a read-only input whose state spaces are closed under simple operations.

Lemma 1. *Any deterministic or non-deterministic class of real-time dynamical recognizers for which the set of allowed f_a is closed under direct product, and for which the set of allowed H_{yes} is closed under direct product and union, is closed under union and intersection.*

Proof. Suppose we have two recognizers ρ_1 and ρ_2 with functions f_a and g_a on spaces M and N and accepting subsets $J_{\text{yes}} \subset M$ and $K_{\text{yes}} \subset N$ respectively. Then define a new recognizer ρ with $h_a = f_a \times g_a$ on $M \times N$; in other words, simply run both recognizers in parallel.

Then to recognize $L_{\rho_1} \cap L_{\rho_2}$ or $L_{\rho_1} \cup L_{\rho_2}$, let $H_{\text{yes}} = J_{\text{yes}} \times K_{\text{yes}}$ or $H_{\text{yes}} = (J_{\text{yes}} \times N) \cup (M \times K_{\text{yes}})$ respectively. ■

This includes all of the recognizer classes under discussion.

Lemma 2. *Any deterministic class of recognizers for which the set of allowed H_{yes} is closed under complementation is closed under complementation.*

Proof. Let $H'_{\text{yes}} = \overline{H_{\text{yes}}}$. ■

This includes all of the deterministic classes under discussion. It doesn't work for non-deterministic ones, since the complement of a non-deterministic language is the set of words for which all computation paths reject, namely a set defined by a $\forall$ quantifier ("for all") rather than a $\exists$ ("there exists"). This is typically not another non-deterministic language.

A *homomorphism* from one language to another is a map h from its alphabet to the set of finite words in some (possibly different) alphabet. For instance, if $h(a) = b$ and $h(b) = ab$, then $h(bab) = abbab$. If L is a language, then its image and inverse image under h are $h(L) = \{h(w) \mid w \in L\}$ and $h^{-1}(L) = \{w \mid h(w) \in L\}$.

A homomorphism is *ϵ-free* if no symbol is mapped to the empty word, and *alphabetic* if each symbol is mapped to a one-symbol word.

Lemma 3. *Deterministic and non-deterministic recognizer classes for which the set of allowed f_a is closed under composition are closed under inverse homomorphism. All recognizer classes are closed under alphabetic inverse homomorphism.*

Proof. If we have a recognizer ρ for a language L, we can make a recognizer for $h^{-1}(L)$ by converting the input word w to $h(w)$ and feeding $h(w)$ to ρ. To do this, simply replace f_a with $f_{h(a)}$ (where f_ϵ is the identity function ι), i.e. just compose the maps for the symbols in $h(a)$. If the homomorphism is alphabetic, $h(a)$ is a single symbol and no composition of functions is necessary. ■

Since linear, polynomial and piecewise-linear functions are closed under composition, we have

Corollary. $\mathbf{Lin}, \mathbf{NLin}, \mathbf{Poly}, \mathbf{NPoly}, \mathbf{PieceLin}$ *and* $\mathbf{NPieceLin}$ *are closed under inverse homomorphism.*

We actually mean $\mathbf{Lin}(U), \mathbf{NLin}(U), \mathbf{Poly}(U)$ and so on are each closed under h^{-1} for $U = \mathbb{Z}$, $\mathbb{Q}$ or $\mathbb{R}$. These are potentially distinct classes (although see theorem 3).

Lemma 4. *Any non-deterministic language is an alphabetic homomorphism of a language in the corresponding deterministic class.*

Proof. If each symbol a has several choices of map $f_a^{(i)}$, make the recognizer deterministic by expanding the alphabet to $\{(a,i)\}$ so that the input explicitly tells it which map to use. Then $h((a,i)) = a$ is an alphabetic homomorphism. ■

Lemma 5. *FSAs with n states can be simulated by linear maps in n dimensions.*

Proof. Simply use the unit vectors $e_i = (0, \ldots, 1, \ldots, 0)$ to represent the states of a FSA, with f_a acting as the transition matrix when it reads the symbol a. Then let x_0 be the e_i corresponding to the start state, and let H_{yes} pick out the e_i corresponding to accepting final states. (Deterministic maps suffice, since non-deterministic and deterministic finite state automata can both recognize the regular languages [21].) ■

Corollary. $\mathbf{Reg} \subset \mathbf{Lin}(\mathbb{Z})$.

This containment is proper, since the example $\{w \mid \#_a(w) \geq \#_b(w)\}$ given in the introduction is a non-regular language.

Lemma 6. *Non-deterministic recognizer classes containing linear maps are closed under ϵ-free homomorphism.*

Proof. We have to show that a recognizer ρ for a language L can be converted into one ρ' for $h(L)$. Specifically, ρ' will work by guessing a pre-image $h^{-1}(w)$ of the input word, and applying ρ to that pre-image.

Consider a non-deterministic FSA with states labelled (a, i), representing a guess that we are currently reading the ith symbol of $h(a)$ where a is a symbol in the pre-image of w. Add a start state I and a reject state R. Let it make transitions based on the current input symbol u in the obvious way:

$$\begin{array}{rcl}
I & \to & (a,1) \text{ if } u = h(a)_1 \\
(a,i) & \to & \left\{ \begin{array}{l} (a,i+1) \text{ if } u = h(a)_{i+1} \\ R \text{ otherwise} \end{array} \right. \\
(a,|h(a)|) & \to & (b,1) \text{ if } u = h(b)_1 \\
R & \to & R
\end{array}$$

In order to plug the original recognizer ρ into this FSA, we apply f_a to x whenever we complete a word $h(a)$, i.e. when the FSA arrives at the state $(a, |h(a)|)$, and leave x unchanged otherwise. We next show how to do this.

Suppose ρ acts on a space M. Then let the new recognizer ρ' act on $M' = M^n$ where n is the total number of states in the FSA. At all times, the state $x' \in M'$ will be a vector with only one non-zero component $x'_s = x$, where s is the current FSA state and x is the simulated state of ρ. Denote this vector x^s.

Then for each symbol a and each allowed transition $s \to t$ of the FSA, define a non-deterministic map

$$f_{a,s}^{(t)} = \begin{cases} f_a \text{ if } t = (a, |h(a)|) \\ \iota \text{ otherwise} \end{cases}$$

where ι is the identity function. Then let f'_a be the non-deterministic map

$$f_a'^{(\{t_s\})}(x') = \sum_s (f_{a,s}^{(t_s)}(x'_s))^{t_s}$$

where we non-deterministically choose a transition $s \to t_s$ for each s. Finally, let $x'_0 = x_0^I$ and let $H'_{\text{yes}} = \cup_a H_{\text{yes}}^{(a,|h(a)|)}$ so that we accept only when we have completed the last symbol in the pre-image and x is in H_{yes}. ■

Non-determinism is required here in general, since most homomorphisms are many-to-one. However, deterministic maps suffice for one-to-one (or constant-to-one) homomorphisms where we only need to look ahead a constant number of symbols to determine the pre-image, such as the $h(a) = b$, $h(b) = ab$ example above.

Recall [21] that a *trio* is a class of languages closed under inverse homomorphism, ϵ-free homomorphism, and intersection with a regular language. (For a formal treatment of trios and other families of languages closed under various operations, see [2].) Then we have shown that

Theorem 1. $\mathbf{NLin}, \mathbf{NPoly}$ *and* $\mathbf{NPieceLin}$ *are trios.*

Proof. Lemma 3 applies since all these classes are closed under composition. Lemmas 1, 5, and 6 also apply. ■

The *interleave* of two languages $L_1 \wr L_2$ is the set of words

$$\{w_1 x_1 w_2 x_2 \cdots w_k x_k \,|\, w_1 w_2 \cdots w_k \in L_1, x_1 x_2 \cdots x_k \in L_2\}$$

where the w_i and x_i are words, including possibly ϵ. For instance, $\{ab\} \wr \{cd\} = \{abcd, acbd, acdb, cabd, cadb, cdab\}$. The *concatenation* of two languages $L_1 \cdot L_2$ is the set of words $\{wx \,|\, w \in L_1, x \in L_2\}$. Then

Lemma 7. *Non-deterministic classes closed under direct product are closed under interleaving, and non-deterministic classes that include linear maps are closed under concatenation. Deterministic classes are closed under these operations if L_1 and L_2 have disjoint alphabets.*

Proof. Suppose L_1 and L_2 are recognized by ρ_1 and ρ_2 with maps g_a and h_a on spaces M and N, with initial points y_0 and z_0 and accepting subsets J_{yes} and K_{yes} respectively.

Then $L_1 \wr L_2$ is recognized by ρ on $M \times N$ with $x_0 = (y_0, z_0)$, $H_{\text{yes}} = J_{\text{yes}} \times K_{\text{yes}}$, and where f_a non-deterministically chooses between $g_a \times \iota$ and $\iota \times h_a$; in other words, with each symbol we update either ρ_1 or ρ_2, and we demand that both reach an accepting state by the end. If L_1 and L_2 have disjoint alphabets, there is no ambiguity about which map to apply and deterministic maps suffice.

To recognize $L_1 \cdot L_2$, expand the space to $M \times N \times \mathbb{R}^2$, and let $x_0 = (y_0, z_0, 1, 0)$. We can use the last two components as a finite-state machine to enforce that we never follow a map from ρ_2 by one from ρ_1 by letting $f_a(y, z, s, t)$ choose between

$$(g_a(y), z, s, 0) \text{ and } (y, h_a(z), 0, s+t)$$

Then if we ever follow the second map with the first, both s and t will be zero. So let

$$H_{\text{yes}} = J_{\text{yes}} \times K_{\text{yes}} \times \overline{(0,0)}$$

More abstractly, we can use an alphabetic inverse homomorphism h^{-1} to send L_1 and L_2 to disjoint alphabets A_1 and A_2. Then

$$L_1 \cdot L_2 = h\left((h^{-1}(L_1) \wr h^{-1}(L_2)) \cap (A_1 \cdot A_2)\right)$$

is in the class by lemmas 1, 3, 5, and 6 (since $A_1 \cdot A_2$ is a regular language). If L_1 and L_2 already have disjoint alphabets, then no homomorphism is necessary:

$$L_1 \cdot L_2 = (L_1 \wr L_2) \cap (A_1 \cdot A_2)$$

and deterministic maps suffice by lemmas 1 and 5. ∎

This ability to run several recognizers in parallel gives dynamical recognizers some closure properties that not all trios have. For instance, the context-free languages are not closed under interleaving or intersection.

Now let a *=0-recognizer* be one for which $H_{\text{yes}} = \{x \mid h(x) = 0\}$ for some h, and call a class of such recognizers a *$=0$-class*. Define >0 and ≥ 0-classes similarly. Write subsets of the classes we've already defined as $\mathbf{NPoly}_{=0}$, $\mathbf{NPieceLin}_{\geq 0}$, and so on. Then

Lemma 8. *For* **PieceLin** *and* **NPieceLin**, *the =0-classes and ≥0-classes coincide.*

Proof. Let $f(x) = |x| - x$ and $g(x) = -|x|$. Then $f(x) = 0$ if and only if $x \geq 0$, and $g(x) \geq 0$ if and only if $x = 0$. Then if h is a measurement function in the =0-class (resp. ≥0-class) then $g \circ h$ ($f \circ h$) is in the ≥0-class (=0-class). ∎

As alluded to in the definition of regular languages above, the *Kleene star* of a language L consists of zero or more concatenations of it, $L^* = \cup_{i \geq 0} L^i = \epsilon + L + (L \cdot L) + \cdots$. The *positive closure* of a language L is $L^+ = \cup_{i \geq 1} L^i$, one or more concatenations.

Lemma 9. *Non-deterministic =0-classes that are closed under composition, and that contain a function $f_\wedge$ such that $f_\wedge(x, y) = 0$ if and only if $x = y = 0$, are closed under positive closure and Kleene star. Similarly for >0-classes and ≥0-classes.*

Proof. For the =0-classes, let $M' = M \times \mathbb{R}$ with $x_0' = (x_0, 0)$. Then define $f_a'(x, y)$ non-deterministically,

$$f_a'(x, y) = \begin{cases} (f_a(x), y) \\ (f_a(x_0), f_\wedge(h(x), y)) \end{cases}$$

That is, either iterate f_a on x or transfer $h(x)$ to y and start over with x_0. Let $h'(x,y) = f_\wedge(h(x), y)$. Then if $w = w_1 w_2 \cdots w_k$,

$$h'(x_w, y_w) = f_\wedge\left(h(x_{w_k}), f_\wedge\left(h(x_{w_{k-1}}), \cdots f_\wedge(h(x_1), 0)\right)\right)$$

and $h' = 0$ if and only if $h(w_i) = 0$ for all i, so all the w_i are in L.

As in lemma 6, non-determinism is required to guess how to parse the input into subwords, unless there is some way of determining this with a bounded look-ahead (such as a symbol that only occurs at the beginning of each word).

If the empty word ϵ is a member of L, then $L^+ = L^*$. If not, add a variable z with $z_0 = 1$ and $f_a(z) = 0$ for all a and let

$$H_{\text{yes}} = \{(x,y,z) \mid h'(x,y) = 0 \text{ or } z = 1\}$$

Then H_{yes} accepts $L^+ \cup \{\epsilon\} = L^*$. Similarly for >0 and ≥0-classes. ■

Finally, recall [21] that an *abstract family of languages* (AFL) is a trio which is also closed under union, concatenation, and positive closure.

Theorem 2. $\mathbf{NPoly}_{=0}$*,* $\mathbf{NPieceLin}_{=0}$ *and* $\mathbf{NPieceLin}_{>0}$ *are AFLs.*

Proof. For $\mathbf{NPoly}_{=0}$, $\mathbf{NPieceLin}_{=0}$ and $\mathbf{NPieceLin}_{>0}$, let $f_\wedge(x,y) = x^2 + y^2$, $|x| + |y|$ and $\min(x,y)$ respectively. Since all these classes are closed under composition, by lemma 9 they are closed under positive closure. We now show that they are also trios, and closed under union and concatenation.

We already have an 'and' function; we need an 'or'. For =0-classes, $f_\vee(x,y) = xy$ is polynomial and $f_\vee(x,y) = \min(|x|,|y|)$ is piecewise-linear. For $\mathbf{NPieceLin}_{>0}$, let $f_\vee(x,y) = x + y + |x| + |y|$.

Then letting $h = f_\wedge(h_1,h_2)$ or $f_\vee(h_1,h_2)$ will recognize $L_1 \cap L_2$ or $L_1 \cup L_2$ respectively. So lemma 1 applies, and we have closure under union and intersection.

Lemmas 3, 6 and 7 also apply since these classes contain the regular languages (inequalities of any kind can be used in lemma 5). This completes the proof. ■

Theorems 1 and 2 suggest that these dynamical classes deserve to be thought of as 'natural' language classes.

5 Linear and polynomial recognizers

We now prove some specific theorems about the linear, piecewise-linear and polynomial language classes. First, we show that rational coefficients are no more powerful than integer ones:

Theorem 3. $\mathcal{C}(\mathbb{Z}) = \mathcal{C}(\mathbb{Q})$ *for* $\mathcal{C} = \mathbf{Poly}_k$, $\mathbf{NPoly}_k$, $\mathbf{PieceLin}$, *and* $\mathbf{NPieceLin}$.

Proof. Suppose a recognizer ρ uses polynomial maps f_a and h of degree k with rational coefficients. We will transform these to maps of the same degree with integer coefficients by using a continually expanding system of coordinates (and one additional variable).

If h and the f_a have rational coefficients, then there exists a q such that qh and $qf_a(x)$ have integer coefficients. If $f_a(x) = c_k x^k + \cdots + c_0$, add a variable r with $r_0 = 1$ and let

$$g_a(x,r) = qc_k x^k + qc_{k-1}x^{k-1}r + \cdots + qc_1 x r^{k-1} + qc_0 r^k = qr^k f_a(x/r)$$

and

$$f'_a(x,r) = (g_a(x,r), qr^k)$$

Then the reader can easily check that

$$x'_w = f'_w(x_0, r_0) = (r_t x_w, r_t)$$

where

$$r_t = q^{k^{t-1}+\cdots+k+1}$$

Finally, if one of the inequalities in H_{yes} is $h(x) > 0$, let

$$h'(x,r) = qr^k h(x/r)$$

so that $h'(x'_w) = qr_t^k h(x_w)$, and $h' > 0$ iff $h > 0$. Similarly for $h = 0$ or $h \geq 0$.

Then f'_a and h' are polynomials of degree k with integer coefficients. We can easily transform the coefficients and component boundaries of piecewise-linear maps in the same way. ■

Henceforth we will simply refer to $\mathbf{Lin}(\mathbb{Z})$, $\mathbf{Poly}(\mathbb{Z})$, etc.

5.1 Queues and stacks

We now explore the specific abilities of the first few classes. A *k-tape real-time queue automaton* [9] is a finite-state machine with access to k queues. The queues are first-in-first-out (FIFO), so that the machine can add symbols at one end (say the right), but only read them at the other (say the left). At each time-step the machine reads a symbol of the input word and, based on this and the leftmost symbol in each queue, it may 1) add a finite word to each queue, 2) pop the leftmost symbol off one or more queues, and 3) update its own state. The machine accepts a word if its FSA ends in an accepting state. The languages recognized by deterministic and non-deterministic k-queue automata are called $\mathbf{QA}_k$ and $\mathbf{NQA}_k$ respectively, and $\mathbf{QA} = \cup_k \mathbf{QA}_k$ and $\mathbf{NQA} = \cup_k \mathbf{NQA}_k$.

Here we will add a new class $\mathbf{CQA} \subseteq \mathbf{QA}$, the languages recognized by *copy queue automata*. Instead of popping symbols off a queue q, CQAs push them onto a 'copy queue' q' and demand at the end of the computation that $q' = q$ (or inequalities such as $q' \sqsubseteq q$, i.e. q' is an initial subsequence of q). Equivalently, CQAs allow us to pop symbols we haven't pushed yet, as long as we push them before we're done. If you like, it creates 'ghost symbols' that haunt the queue until they are cancelled by pushing real ones. CQAs can be deterministic or non-deterministic (NCQAs).

Finally, we say a deterministic QA or CQA is *obstinate* if its move, including what symbols if any it wants to push or pop, depends only on the input symbol and the FSA state, and not on any of the queue symbols. If the symbols it wants to pop aren't there, it rejects immediately. For instance, the copy language $L_{\text{copy}} = \{waw\}$, of words repeated twice with a marker a in the middle, is in the class **OCQA**.

Then we can show

Theorem 4. *The following containments hold, and are proper:*

$$
\begin{aligned}
&\mathbf{NQA} \subset \mathbf{NPieceLin}(\mathbb{Z}) \cap \mathbf{NPoly}_2(\mathbb{Z}) \\
&\mathbf{QA} \subset \mathbf{PieceLin}(\mathbb{Z}) \cap \mathbf{NPoly}_2(\mathbb{Z}) \\
&\mathbf{OQA} \subset \mathbf{PieceLin}(\mathbb{Z}) \cap \mathbf{Poly}_2(\mathbb{Z}) \\
&\mathbf{NCQA} \subset \mathbf{NLin}(\mathbb{Z}) \\
&\mathbf{OCQA} \subset \mathbf{Lin}(\mathbb{Z})
\end{aligned}
$$

Proof. Let the queue alphabet be $\{1, 2, \ldots, m\}$. Then we will represent a word w we wish to push or pop by a real number $\overline{w} = \sum_{i=1}^{|w|} w_i(m+1)^{-i} = .w_1 w_2 \ldots w_{|w|}$ in base $m+1$.

Each queue will be represented by the digit sequence of a variable q, with a 'pointer' $r = (m+1)^{-k}$ where k is the number of symbols in the queue. Let $q_0 = 0$ and $r_0 = 1$. Then the functions

$$
\begin{aligned}
\text{push}_w^{\text{right}}(q, r) &= (q + \overline{w} r,\ (m+1)^{-|w|} r) \\
\text{pop}_w^{\text{left}}(q, r) &= \Big((m+1)^{|w|}(q - \overline{w}),\ (m+1)^{|w|} r\Big)
\end{aligned}
$$

push w onto the least significant digits, pop w off the most significant, and update r accordingly. Since for any particular QA the $\overline{w}$ are constants, these maps are linear.

It is easy to see that the final value of each q will be within the unit interval if and only if the sequence of symbols we popped off each queue is an initial subsequence of the symbols we pushed. Therefore, we add $q_i \in [0, 1)$ for all queues $1 \leq i \leq k$ as an accepting condition along with the final state of the FSA.

However, unless we're dealing with a CQA, we also need to make sure that $q_i \in [0, 1)$ throughout the computation, i.e. we don't pop symbols off before we push them. We can ensure this either with piecewise-linear maps that sense when q falls outside the unit interval, or with a variable s for each queue with $s_0 = 1$ and a quadratic map $f(s) = s(r+1)$ such that $s = 0$ if the r ever becomes -1 during the computation, i.e. if we pop more symbols than we've pushed. Then we add $s \neq 0$ for each queue as an accepting condition.

For a CQA, we require that $q = q'$, or that $q - q' \in [0, r_{q'})$ if we wish q' to be an initial subsequence of q.

And unless our automaton is obstinate, we need to sense the most significant digits of q. Piecewise-linear maps can do this, so that (deterministic) QAs can be simulated by (deterministic) piecewise-linear maps; but linear or quadratic maps seem to be too smooth for this, so they will have to non-deterministically guess the most significant digit even if the QA is deterministic.

In other words, $\mathbf{OCQA} \subset \mathbf{Lin}(\mathbb{Z})$. Relaxing 'copyness' requires piecewise-linear or quadratic maps, relaxing obstinacy requires piecewise-linear maps or non-determinism, and non-determinism requires non-determinism. From these observations follow the containments stated above.

To show that these containments are proper, consider the language of palindromes $L_{\text{pal}} = \{waw^R\}$, where w^R means w in reverse order (we assume w is in an alphabet not including a). This language is known [7] not to be in **NQA**; we will show it is in $\mathbf{Lin}(\mathbb{Z})$.

By using $\text{push}_w^{\text{left}} = (\text{pop}_w^{\text{left}})^{-1}$, we can push symbols on to the left end of the queue, i.e. the most significant digits of q. Do this for the first copy of w until you see the a, whereupon switch (with a FSA control as in Lemma 6) to $\text{pop}_w^{\text{left}}$ and remove the symbols as they appear in reverse order. Then accept if $q = 0$ at the end. So L_{pal} is in $\mathbf{Lin}(\mathbb{Z})$ but not in **NQA**. ■

To recognize L_{pal}, we used the digits of q as a stack rather than a queue. With this construction, we can recognize a subset of the context-free languages. A *metalinear language* [21] is one accepted by a PDA which makes a bounded number of *turns*, a turn being a place in its computation where it switches from pushing to popping. We can also consider obstinate PDAs, which like obstinate QAs only look at the stack symbol to see if it's the one they wanted to pop.

Let the (deterministic, obstinate) metalinear languages be **Met**, **DMet** and **OMet**; for instance, L_{pal} is in **OMet**. (Incidentally, **Met** is a trio). Then

Theorem 5. *The following containments hold, and are proper:*

$$\begin{aligned} &\mathbf{Met} \subset \mathbf{NLin}(\mathbb{Z}) \\ &\mathbf{DMet} \subset \mathbf{PieceLin}(\mathbb{Z}) \\ &\mathbf{OMet} \subset \mathbf{Lin}(\mathbb{Z}) \end{aligned}$$

Proof. Simulate a PDA with $\text{push}_w^{\text{left}}$ and $\text{pop}_w^{\text{left}}$ as we did above, pushing and popping the most significant digits of q. To make sure we pop what we push, it suffices to check that $q \in [0, 1)$ each time the PDA turns from popping to pushing (there are $k - 1$ of these 'interior turns') as well as at the end of the computation. Otherwise, we risk sequences like

$$\begin{array}{rl} & q_0 = 0 \\ \text{push 1:} & q = 0.1 \\ \text{pop 2:} & q = -1 \end{array}$$

$$\text{push } 2: \qquad q = 0.1$$
$$\text{pop } 1: \qquad q = 0$$

where we popped a 2 when the stack symbol was a 1, and then covered our tracks by pushing it again. Here q ends at 0, but at the turn from popping to pushing, $q = -1$ and we fell outside the unit interval.

To prevent this, copy q into a storage variable s_i each time the PDA makes an interior turn. If it turns k times, then only k such variables are needed; so to accept, demand that $s_i \in [0, 1)$ for all i (and that the FSA be in an accepting final state).

As before, piecewise-linear maps can be deterministic if the PDA is, while linear maps have to non-deterministically guess what symbol to pop, unless the PDA is obstinate.

To show that these containments are proper, we note that the language of words with more a's than b's is not metalinear [21], while we showed in the introduction that it is in $\mathbf{Lin}(\mathbb{Z})$. ■

In addition to **OMet**, some context-free languages that are not metalinear are in $\mathbf{Lin}(\mathbb{Z})$, such as the language $\{w \mid \#_a(w) \geq \#_b(w)\}$ of the introduction. This seems to be because when its PDA tries to pop a symbol off an empty stack, it starts a 'negative stack' rather than rejecting; for instance, if we represent extra a's by having a's on the stack, popping one off each time we read a b, we can simply start putting b's on the stack instead if we run out of a's. This is reminiscent of the copy queue automata above, and presumably $\mathbf{Lin}(\mathbb{Z})$ contains some suitably defined subset of the CFLs where 'ghost symbols' can be popped off the stack and later pushed into a peaceful grave.

The Dyck language $\{\epsilon, (), (()), ()(), \ldots\}$, however, relies on rejecting if the stack is overdrawn. We conjecture that

Conjecture 1. *L_{Dyck} is not in* $\mathbf{Lin}(\mathbb{Z})$.

Proof? For a language L, say that w *pumps* L if $uwv \in L$ if and only if $uv \in L$, i.e. inserting or removing w doesn't change whether a word is in L or not. Let $\sigma(w) = w_2 w_3 \cdots w_{|w|} w_1$. Then the conjecture would follow if for any L in $\mathbf{Lin}(\mathbb{Z})$, whenever w pumps L, then $\sigma(w)$ does also.

The proof of this might go like this: if w pumps L, then $f_w \sim \iota$ where "$\sim$" means some sort of equivalence on the subset of $\mathbb{R}^d$ generated by the f_a. Then

$$f_{\sigma(w)} = f_{w_1} f_{w_{|w|}} \cdots f_{w_3} f_{w_2} = f_{w_1} \circ f_w \circ f_{w_1}^{-1} \sim f_{w_1} \circ \iota \circ f_{w_1}^{-1} = \iota$$

Since "()" pumps L_{Dyck}, we would have $f_(\sim f_)^{-1}$ and $f_{()} \sim f_{)(} \sim \iota$. Then ")(" would pump L_{Dyck} also, which it doesn't.

In any case, we can keep track of a stack with an unbounded number of turns with a little more work. Let **OCF**, the *obstinate context-free languages*, be the languages recognized by obstinate PDAs.

Theorem 6. *The following containments hold:*

$$\mathbf{CF} \subset \mathbf{NPieceLin}(\mathbb{Z}) \cap \mathbf{NPoly}_2(\mathbb{Z})$$

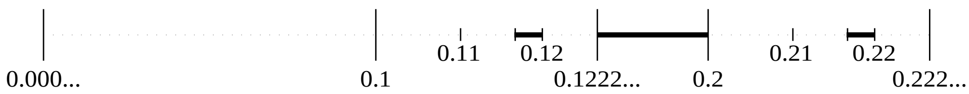

Figure 2: The Cantor set encoding of words on alphabets with m symbols. Here $m = 2$ and $\alpha = 1/4$.

$$\mathbf{DCF} \subset \mathbf{PieceLin}(\mathbb{Z}) \cap \mathbf{NPoly}_2(\mathbb{Z})$$
$$\mathbf{OCF} \subset \mathbf{PieceLin}(\mathbb{Z}) \cap \mathbf{Poly}_2(\mathbb{Z})$$

Proof. To recognize arbitrary context-free languages, we need to make sure that the stack variable q is in the unit interval at all times. To do this quadratically, note that the map

$$f_\wedge(x, y) = (x^2 + y^2)/2$$

has the property that if $x, y \in [0, 1]$ then $f_\wedge(x, y) \in [0, 1]$, while if $|x| \geq 2$ or $|y| \geq 2$ then $f_\wedge(x, y) \geq 2$. So rather than simply using base $m + 1$, we will use a Cantor set with gaps between the blocks. If the gaps are large enough, any mistake will send q far enough outside $[0, 1]$ that $f_\wedge$ will be able to sense the mistake and remember it.

To push and pop single symbols $1 \leq i \leq m$ in the stack alphabet, let

$$\begin{aligned} \text{push}_i(q) &= \alpha q + (1 - \alpha)\frac{i}{m} \\ \text{pop}_i(q) &= \alpha^{-1}\left(q - (1 - \alpha)\frac{i}{m}\right) = \text{push}_i^{-1}(q) \end{aligned}$$

If $\alpha = 1/(m + 1)$ this is just $\text{push}^{\text{left}}$ and pop^{left} in base $m + 1$ as before; smaller α gives a Cantor set as shown in figure 2.

If we pop the wrong symbol, our value of q will be

$$q_{\text{oops}} = \text{pop}_j(\text{push}_i(q)) = q + \left(\frac{1 - \alpha}{\alpha}\right)\left(\frac{i - j}{m}\right)$$

where $i \neq j$, then

$$|q_{\text{oops}}| \geq \frac{1 - \alpha}{m\alpha} - 1$$

Then if we choose α such that $\alpha \leq 1/(3m + 1)$, any mistake will result in $|q_{\text{oops}}| \geq 2$.

Then as in lemma 9, add a variable y with $y_0 = 0$ and update it to $f_\wedge(q, y)$ at each step. Then requiring $y \in [0, 1]$ in H_{yes} ensures that $|q|$ has always been less than 2, i.e. we always popped symbols that were actually there. With piecewise-linear maps, we can use $f_\wedge(q, y) = \max(|q|, |y|)$.

Once again, (deterministic) piecewise-linear maps can read the top stack symbol of (deterministic) PDAs, while for non-obstinate PDAs quadratic maps need to guess. ■

The fact that **CF** $\subset$ **NPieceLin**$(\mathbb{Z})$ and **DCF** $\subset$ **PieceLin**$(\mathbb{Z})$ was essentially shown in [31] and [1].

The closure of **CF** (**DCF**, **OCF**) under intersection and union is the class of *concurrent (deterministic, obstinate) context-free languages*, or **CCF** (**CDCF**, **COCF**). They are recognized by (deterministic, obstinate) PDAs with access to any finite number of stacks. Then

Corollary 1. *The following containments hold, and are proper:*

$$\begin{aligned} \mathbf{CCF} &\subset \mathbf{NPieceLin}(\mathbb{Z}) \cap \mathbf{NPoly}_2(\mathbb{Z}) \\ \mathbf{CDCF} &\subset \mathbf{PieceLin}(\mathbb{Z}) \cap \mathbf{NPoly}_2(\mathbb{Z}) \\ \mathbf{COCF} &\subset \mathbf{PieceLin}(\mathbb{Z}) \cap \mathbf{Poly}_2(\mathbb{Z}) \end{aligned}$$

Proof. The containments follow since all the classes in theorem 6 are closed under intersection and union. To show that they are proper, recall that for a context-free language L on a one-symbol alphabet $\{a\}$, the set $\{n \mid a^n \in L\}$ is eventually periodic [21]. Since the intersection or union of eventually periodic sequences is eventually periodic, this holds for **CCF** as well.

But **Lin**$(\mathbb{Z})$ contains numerous non-periodic one-symbol languages. Consider a recognizer ρ with

$$f_a(x,y) = \begin{pmatrix} 2 & -1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

which dilates the x, y plane and rotates it by an irrational angle $\tan^{-1}(1/2)$. Then if $(x_0, y_0) = (1,0)$ and H_{yes} is the upper half-plane, $\{n \mid a^n \in L_\rho\}$ is a quasiperiodic sequence and so $L_\rho \notin$ **CCF**.

Alternately, consider the language $L_{\text{copy}} = \{waw\}$, which is in $\mathbf{QA}_1$ and **Lin**$(\mathbb{Z})$ but not **CCF** (this follows easily from the results in [27]). ■

Corollary 2. $\mathbf{NTIME}(\mathcal{O}(n)) \subseteq \mathbf{NPieceLin}(\mathbb{Z}) \cap \mathbf{NPoly}_2(\mathbb{Z})$ *and* $\mathbf{TIME}(n) \subseteq \mathbf{PieceLin}(\mathbb{Z}) \cap \mathbf{NPoly}_2(\mathbb{Z})$.

Proof. We have shown that (deterministic) piecewise-linear and non-deterministic quadratic maps can simulate (deterministic) FSAs with access to a finite number of strings that can act like both queues and stacks, i.e. that we can read, push or pop at either end. These are called double-ended queues or *deques* in the literature, but I prefer to call them *quacks*.

FSAs with access to a finite number of quacks can simulate multi-tape Turing machines in real time, and vice versa [26]. Book and Greibach [6] showed that in the non-deterministic case, real time (n) is equivalent to linear time $(\mathcal{O}(n))$. Furthermore, $\mathbf{NTIME}(\mathcal{O}(n))$ is precisely the images of languages in **CCF** under alphabetic homomorphisms, and is the smallest AFL containing **CF**. ■

Piecewise-linear and non-deterministic recognizers seem more powerful than Turing machines. For instance, they can compare the contents of two tapes in a single step, or add one tape to another. We therefore conjecture that

Conjecture 2. **PieceLin** *and* $\mathbf{NPoly}_2$ *maps are more powerful than Turing machines in real time, i.e. the inclusions in corollary 2 are proper.*

5.2 A language not in Poly or PieceLin, and its consequences

Our next theorem puts an upper bound on the memory capacity of deterministic piecewise-linear and polynomial maps of any degree.

Theorem 7. *The language*

$$L_7 = \{w_1 \sharp w_2 \sharp \cdots \sharp w_m \natural v \mid v = w_i \text{ for some } i\}$$

where the w_i and v are in A^*, is in $\mathbf{NLin}(\mathbb{Z})$ but not $\mathbf{Poly}(\mathbb{R})$ or $\mathbf{PieceLin}(\mathbb{R})$.

Proof. Note that L_7 is a kind of universal language, in that it can be "programmed" to recognize any finite language: if $u = w_1 \sharp w_2 \sharp \cdots \sharp w_m \natural$ where $w_1, \ldots, w_m$ are all the words in a finite language L_u, then $uv \in L_7$ if and only if $v \in L_u$. Therefore, any recognizer ρ for L_7 contains recognizers for all possible finite languages in its state space, since it recognizes L_u if we let $x_0 = x_u$. We will show that no polynomial recognizer of finite degree can have this property.

A family of sets $S_1, \ldots, S_n$ is *independent* if all 2^n possible intersections of the S_i and their complements are non-empty; in other words, if the S_i overlap in a Venn diagram. But since $f_v(x_u) \in H_{\text{yes}}$ if and only if $v \in L_u$, x_u is in the following intersection of sets:

$$x_u \in \left(\bigcap_{v \in L_u} f_v^{-1}(H_{\text{yes}}) \right) \cap \left(\bigcap_{v \notin L_u} \overline{f_v^{-1}(H_{\text{yes}})} \right)$$

Since any such intersection is therefore non-empty, the set of sets $f_v^{-1}(H_{\text{yes}})$ over any finite set of words w is independent.

Now a theorem of Warren [40] states that m polynomials of degree k can divide $\mathbb{R}^d$ into at most $(4emk/d)^d$ components if $m \geq d$. If this number is less than 2^m, then not all these sets can be independent.

Suppose ρ is polynomial of degree k, has d dimensions, and has an alphabet with n symbols. Assume for the moment that H_{yes} is defined by a single polynomial inequality of degree k. Then $f_v^{-1}(H_{\text{yes}})$ is defined by a polynomial of degree $k^{|v|+1}$. Then for all n^l of the sets $f_v^{-1}(H_{\text{yes}})$ for words of length l to be independent, we need

$$\left(\frac{4en^l k^{l+1}}{d} \right)^d \geq 2^{n^l}$$

This is clearly false for sufficiently large l, since the right-hand side is doubly exponential in l while the left-hand side is only singly so.

If H_{yes} is defined by c inequalities instead of one, we simply replace n^l with cn^l on the left-hand side; the right-hand side remains the same, since we still need to create n^l independent sets.

Thus polynomial maps of a fixed degree, in a fixed number of dimensions, cannot be programmed to recognize arbitrary finite languages of words of arbitrary length, so L_7 is not in $\mathbf{Poly}(\mathbb{R})$. A similar argument works for piecewise-linear maps, as long as the number of components of the map is finite.

However, L_7 is in $\mathbf{NLin}(\mathbb{Z})$: just non-deterministically keep $\overline{w}_i$ for some i and ignore the others, and check that $\overline{v} = \overline{w}_i$. ■

(Another language we could use here is $\{w\natural k \mid w_{\overline{k}} = 1\}$.) Several corollaries follow from theorem 7, using arguments almost identical to those used in [37] for the deterministic real-time languages $\mathbf{TIME}(n)$:

Corollary 1. $\mathbf{Poly}$, $\mathbf{Poly}_k$ *for all* k, *and* $\mathbf{PieceLin}$ *are properly contained in* $\mathbf{NPoly}$, $\mathbf{NPoly}_k$, *and* $\mathbf{NPieceLin}$ *respectively, for both* $U = \mathbb{Z}$ *and* $\mathbb{R}$.

Corollary 2. *There are non-deterministic context-free languages not in* $\mathbf{Poly}(\mathbb{R})$ *or* $\mathbf{PieceLin}(\mathbb{R})$.

Proof. The *reversal* of a word w is $w^R = w_{|w|} \cdots w_2 w_1$. Let L' be a modified version of L_7 in which $v^R = w_i$ for some i, instead of $v = w_i$. Then L' is context-free: it is accepted by a non-deterministic PDA that puts one of the w_i on the stack, ignores the others, and then compares v to it in reverse. However, L' is not in $\mathbf{Poly}(\mathbb{R})$ or $\mathbf{PieceLin}(\mathbb{R})$ by the same argument we used for L_7.

Corollary 3. $\mathbf{Poly}$, $\mathbf{Poly}_k$ *for all* k, *and* $\mathbf{PieceLin}$ *are not closed under alphabetic homomorphism, concatenation, Kleene star or positive closure.*

Proof. By lemma 4, since $L_7 \in \mathbf{NLin}(\mathbb{Z})$, it is an alphabetic homomorphism h of a language in $\mathbf{Lin}(\mathbb{Z})$: simply mark the w_i that v will be equal to, and let h remove the mark. So none of these classes can be closed under h.

For concatenation, let L_1 be a modified version of L_7 where $v = w_1$. Then L_1 is in $\mathbf{Lin}(\mathbb{Z})$, since we can ignore everything between the first $\sharp$ and the $\natural$, and just compare $\overline{v}$ to $\overline{w}_1$. Then $L_7 = (A \cup \{\sharp\})^* \cdot L_1$ is the concatenation of a regular language with L_1, so these classes can't be closed under concatenation (or even concatenation with a regular language).

Finally, $L'' = (A \cup \{\sharp\})^* \cup L_1$ is in $\mathbf{Lin}(\mathbb{Z})$. But

$$L_7 = L''^* \cap (A \cup \{\sharp\})^* \natural A^*$$

Since these classes are closed under intersection, they can't be closed under Kleene star or positive closure (we can use L''^+ in place of L''^*.) ■

Let $\mathrm{CYCLE}(L) = \{w_1 w_2 \mid w_2 w_1 \in L\}$. Then:

Corollary 4. $\mathbf{Poly}$, $\mathbf{Poly}_k$ *for* $k \geq 2$, *and* $\mathbf{PieceLin}$ *are not closed under reversal or CYCLE.*

Proof. L_7^R, where the first word has to be equal to one that follows, is in $\mathbf{Poly}_2(\mathbb{Z})$. Just update a variable y to $y(\overline{v} - \overline{w}_i)$ each time you see a $\sharp$ and require that $y = 0$ or $\overline{v} = \overline{w}_1$ at the end. We can do the same thing with piecewise-linear maps. Since $L_7 = (L_7^R)^R$, these classes can't be closed under reversal.

Let L_7' be L_7 where the symbols of v are in a marked alphabet A', while the w_i are still in A^*. Clearly L_7' is not in $\mathbf{Poly}(\mathbb{R})$ or $\mathbf{NPoly}(\mathbb{R})$ for the same reason that L_7 isn't, while ${L'_7}^R$ is in $\mathbf{Poly}_2(\mathbb{Z})$ and $\mathbf{NPoly}(\mathbb{Z})$ just as L_7^R is. But

$$L'_7 = \mathrm{CYCLE}({L'_7}^R) \cap (A \cup \{\sharp\})^* \natural A'^*$$

Since both these classes contain regular languages and are closed under intersection, they can't be closed under CYCLE. ■

Conjecture 3. $\mathbf{Lin}(\mathbb{Z})$ *is closed under reversal.*

Proof? We can use transposes to reverse the order of matrix multiplication, since $(AB)^T = B^T A^T$. However, it's unclear how to make these matrices take x_0 to points in H_{yes}, rather than the reverse. We also leave as an open problem whether $\mathbf{Lin}(\mathbb{Z})$ is closed under CYCLE.

On the other hand, we have

Theorem 8. *All non-deterministic classes containing* $\mathbf{Lin}(\mathbb{Z})$ *are closed under reversal and CYCLE.*

Proof. Add variables $p_0 = q_0 = 0$. At each step when we read $a = w_i$, make a guess that $w_i^R = a'$ and let

$$f_a^{(a')}(p, q, x) = (\mathrm{push}_a^{\mathrm{left}}(p), \mathrm{push}_{a'}^{\mathrm{right}}(q), f_{a'}(x))$$

where x represents the other variables. Then $p = \overline{w}$ and $q = \overline{w'^R}$ where w' is composed of the guessed symbols a', so require that $p = q$.

Similarly, for CYCLE, let $p_0 = q_0 = r_0 = 0$. Start out with

$$f_a^{(a')}(p, q, r, x) = (\mathrm{push}_a^{\mathrm{left}}(p), \mathrm{push}_{a'}^{\mathrm{left}}(q), r, f_{a'}(x))$$

and non-deterministically switch to

$$f_a^{(a')}(p, q, r, x) = (\mathrm{push}_a^{\mathrm{left}}(p), q/b, \mathrm{push}_{a'}^{\mathrm{left}}(r), f_{a'}(x))$$

where p, q and r are in base b. Then $p = \overline{w}$ and $q + r = \overline{\mathrm{CYCLE}(w')}$, so require that $p = q + r$. ■

Next, we will show that a unary version of L_7 separates **Lin** from **PieceLin** and from $\mathbf{Poly}_2$ (and from their intersection):

Theorem 9. Lin *is properly contained in* $\mathbf{PieceLin} \cap \mathbf{Poly}_2$ *for both* $U = \mathbb{Z}$ *and* $\mathbb{R}$.

Proof. Consider a version of L_7 where the w_i and v are over a one-symbol alphabet:

$$L_{\text{unary}} = \{a^{p_1} \sharp a^{p_2} \sharp \cdots \sharp a^{p_m} \natural a^q \mid q = p_i \text{ for some } i\}$$

Suppose L_{unary} is in $\mathbf{Lin}(\mathbb{R})$. Since f_{a^i} is linear, if H_{yes} is described by c linear inequalities, then each of the sets $f_{a^i}^{-1}(H_{\text{yes}})$ is also. But these all have to be independent by the same argument as in theorem 7, so for $1 \leq i \leq l$, cl linear inequalities have to divide $\mathbb{R}^d$ into at least 2^l components. But for $k = 1$, Warren's inequality becomes

$$\left(\frac{4ecl}{d}\right)^d \geq 2^l$$

which is false for sufficiently large l. So L_{unary} is not in $\mathbf{Lin}(\mathbb{R})$.

However, L_{unary} is in $\mathbf{PieceLin}(\mathbb{Z})$. Let $x_0 = y_0 = 0$ and $r_0 = 1$, with the following dynamics:

$$\begin{aligned} f_a(x, y, r) &= (x, 2y \bmod 2, r/2) \\ f_\sharp(x, y, r) &= \begin{cases} (x + r, x + r, 1) \text{ if } y \in [0, 1) \\ (x, x, 1) \text{ if } y \in [1, 2) \end{cases} \\ f_\natural(x, y, r) &= f_\sharp(x, y, r) \end{aligned}$$

The sequence $a^p\sharp$ adds 2^{-p} to x unless the 2^{-p} digit of x was already 1, i.e. unless $y = 2^p x \bmod 2 \in [1, 2)$. By the time we reach the $\natural$, we have $x = \sum_i 2^{-p_i}$. Then with an additional variable w, let $f_\natural(w) = x$, let $f_a(w) = 2w \bmod 2$, and let H_{yes} require that $w \in [1, 2)$, checking that the 2^{-q} digit of x is 1.

What about L^R_{unary}? It is in $\mathbf{Poly}_2(\mathbb{Z})$ by the same construction as in corollary 4 above. Just let $f_\sharp(y) = y(q - p_i)$ and require that $y = 0$ or $q = p_1$. Similarly, it is in $\mathbf{PieceLin}(\mathbb{Z})$. However, we can show that it is not in $\mathbf{Lin}(\mathbb{R})$.

Let $p[j]$ be the 2^j digit of p in base 2. For a given k, and $0 \le j < k$, let u_j be the word

$$u_j = \prod_{0 \le p < 2^k,\, p[j]=1} (\natural/\sharp) a^p$$

where $\prod$ means concatenation and $(\natural/\sharp)$ means $\sharp$ except for the first one, where it means $\natural$. For instance, for $k = 3$,

$$\begin{aligned} u_0 &= \natural a \sharp a^3 \sharp a^5 \sharp a^7 \\ u_1 &= \natural a^2 \sharp a^3 \sharp a^6 \sharp a^7 \\ u_2 &= \natural a^4 \sharp a^5 \sharp a^6 \sharp a^7 \end{aligned}$$

Now let $S_j = f_{u_j}^{-1}(H_{\text{yes}})$. Since $a^p u_j \in L^R_{\text{unary}}$ if and only if $p[j] = 1$, we have

$$x_{a^p} \in \left(\bigcap_{j | p[j]=1} S_j \right) \cap \left(\bigcap_{j | p[j]=0} \overline{S_j} \right)$$

and the S_j are independent. For instance, $x_{a^5} \in S_0 \cap \overline{S_1} \cap S_2$.

But since f_{u_j} is linear, each of the S_j are described by c linear inequalities if H_{yes} is, so ck linear inequalities have to divide $\mathbb{R}^d$ into at least 2^k components and once again Warren's inequality applies. So L^R_{unary} is in $\mathbf{PieceLin}(\mathbb{Z}) \cap \mathbf{Poly}_2(\mathbb{Z})$, but not $\mathbf{Lin}(\mathbb{R})$. ■

(It is an interesting open question whether L_{unary} is in $\mathbf{Poly}_2(\mathbb{Z})$. For $k = 2$ and $n = 1$, Warren's inequality becomes

$$\left(\frac{4ecl\, 2^{l+1}}{d} \right)^d \ge 2^l$$

and no longer yields a contradiction for large l.)

Using the same techniques, we can sharpen theorem 5:

Theorem 10. *There are deterministic metalinear context-free languages not in* $\mathbf{Lin}(\mathbb{Z})$.

Proof. For two words $u, v \in \{0,1\}^*$, say that $u \subset v$ if $u_i \leq v_i$ for all i, e.g. $01001 \subset 11011$. Then consider the language $L_{\subset\text{pal}} = \{uav \,|\, u^R \subset v\}$. It can be recognized by a deterministic, one-turn PDA: simply push u onto the stack until you see the a, then pop u back off as you read v, checking that $u_i^R \leq v_i$ as you go. So $L_{\subset\text{pal}}$ is in **DMet** (in fact, it is one-turn or *linear* [21]).

Now for a given k and $0 \leq j < k$, let v_j be the word of length 2^k such that $(v_j)_i = i[j]$ (here we're numbering v_j's symbols from 0 to $2^k - 1$ instead of from 1 to $|v_j|$ as before). For instance, for $k = 3$,

$$\begin{aligned} v_0 &= 01010101 \\ v_1 &= 00110011 \\ v_2 &= 00001111 \end{aligned}$$

analogous to the u_j of theorem 9. Then if u_m is the word of length 2^k such that its mth symbol is 1 and all its other symbols are 0, we have

$$x_{u_m^R a} \in \left(\bigcap_{j \,|\, m[j]=1} f_{v_j}^{-1}(H_{\text{yes}}) \right) \cap \left(\bigcap_{j \,|\, m[j]=0} \overline{f_{v_j}^{-1}(H_{\text{yes}})} \right)$$

and once again we have an arbitrarily large number of independent sets. So $L_{\subset\text{pal}}$ is not in $\mathbf{Lin}(\mathbb{Z})$. ■

There is an interesting connection between theorems 7, 9 and 10 and computational learning theory. The *Vapnik-Chervonenkis dimension* has been used to quantify the difficulty of learning sets by example [4]. For a family of sets F, the VC dimension is the size of the largest independent family $S \subset \mathcal{F}$. Then our arguments about independent sets can be re-stated in the following way: in $\mathbb{R}^d$ with d fixed, the VC dimension of the family

$$\left\{ f_w^{-1}(H_{\text{yes}}) \,\middle|\, |w| \leq l \right\}$$

is $\mathcal{O}(l)$ for polynomial maps and fixed for linear maps (this also follows from the results in [17]), while L_7 and L_{unary} require a VC dimension of at least n^l and l respectively.

Unfortunately, arguments about independent sets don't seem capable of proving our conjecture that $L_{\text{Dyck}} \notin \mathbf{Lin}(\mathbb{Z})$, since $uv \in L_{\text{Dyck}}$ if u and v have the same non-negative number of excess (s and)s respectively (and Dyck languages with k types of brackets have no more independent sets than palindromes $\{waw^R\}$ where w is over a k-symbol alphabet).

5.3 Discrete time and space complexity

We now compare polynomial recognizers directly to Turing machines.

Theorem 11. *For all $k \geq 2$,*

$$\mathbf{Poly}_k(\mathbb{Z}) \subset \mathbf{TIME}(k^n n \log n)$$
$$\mathbf{NPoly}_k(\mathbb{Z}) \subseteq \mathbf{NTIME}(k^n n \log n)$$

where $\subset$ indicates proper inclusion.

Proof. Calculating a polynomial function consists of a fixed number of multiplications and additions. Multi-tape Turing machines can add m-digit numbers in time $\mathcal{O}(m)$ and multiply them in $\mathcal{O}(m \log m \log\log m)$ [22]. The number of digits of the result is $\mathcal{O}(km)$. Iterating such a polynomial n times on initial values x_0 with a fixed number of digits, then, gives us $\mathcal{O}(k^n)$ digits and a total time $\mathcal{O}(k^n n \log n)$. ■

As in [34] let $\mathbf{E}$ be the class $\cup_k \mathbf{TIME}(k^n)$ of problems solvable in exponential time. Then

Corollary. $\mathbf{Poly}(\mathbb{Z}) \subset \mathbf{E}$ *and* $\mathbf{NPoly}(\mathbb{Z}) \subseteq \mathbf{NE}$.

Since exponential time is rather powerful, this doesn't tell us very much (but we give tighter bounds below). In addition, for particular subsets of **Poly** and **NPoly**, we can say more. Recall [21] the *deterministic context-sensitive languages* $\mathbf{DCS} = \mathbf{SPACE}(n)$. This class is believed, but not known, to be a proper subset of $\mathbf{CS} = \mathbf{NSPACE}(n)$.

Now define a language in **Poly** or **NPoly** as *compact and exponentially bounded*, in $\mathbf{Poly}_{\text{ceb}}$ or $\mathbf{NPoly}_{\text{ceb}}$, if its recognizer acts on a compact subset of $\mathbb{R}^d$ and the final points x_w are bounded away from the boundary of H_{yes} by at least $r^{|w|}$ for some $r < 1$. Then we have

Theorem 12. *The following containments hold:*

$$\mathbf{Poly}(\mathbb{Z})_{\text{ceb}} \subset \mathbf{DCS} \cap \mathbf{TIME}(n^2 \log n \log\log n)$$
$$\mathbf{NPoly}(\mathbb{Z})_{\text{ceb}} \subseteq \mathbf{CS} \cap \mathbf{NTIME}(n^2 \log n \log\log n)$$
$$\mathbf{Lin}(\mathbb{Z}), \mathbf{PieceLin}(\mathbb{Z}) \subset \mathbf{DCS} \cap \mathbf{TIME}(n^2)$$
$$\mathbf{NLin}(\mathbb{Z}), \mathbf{NPieceLin}(\mathbb{Z}) \subseteq \mathbf{CS} \cap \mathbf{NTIME}(n^2)$$

where $\subset$ indicates proper inclusion.

Proof. Since the recognizer's space is compact, we can re-scale the system and assume that x never leaves the unit cube. Then if $r \leq 2^{-b}$, we only need to know b digits of x_w to tell if it is in H_{yes} or not.

Furthermore, since the f_a are polynomials, their derivatives are bounded, say by 2^c. To get m digits of $f_a(x)$, then, we need at most $m + c$ digits of x. In the course of iterating the system n times, then, we never need to know x to more than $b + nc$ digits of accuracy, so we only use an amount of space linear in n.

Again, a multi-tape Turing machine can multiply $\mathcal{O}(n)$ digits in $\mathcal{O}(n \log n \log\log n)$ time, and doing this n times gives us the stated result (or actually something slightly stronger, namely that a single algorithm exists within both the time and space bounds).

Similarly, n iterations of a linear or piecewise-linear function with rational coefficients generates $\mathcal{O}(n)$ digits. Each iteration takes $\mathcal{O}(n)$ time, so n of them take $\mathcal{O}(n^2)$ time.

All these inclusions are proper in the deterministic case, since (we state without proof) L_7 is in both **DCS** and $\mathbf{TIME}(n^2)$. ■

Finally, we note that deterministic linear recognition can be parallelized:

Theorem 13. $\mathbf{Lin}(\mathbb{Z})$ is properly contained in $\mathbf{NC}_2$.

Proof. Any f_a in a linear recognizer can be written $f_a(x) = A_a x + B_a$, where A_a is a matrix and B_a a vector. The composition of two such functions is another of the same form. In fact, by adding an additional dimension we can write $f_a(x, 1) = C_a \begin{pmatrix} x \\ 1 \end{pmatrix}$ where $C_a = \left(\begin{array}{c|c} A_a & B_a \\ \hline 0 & 1 \end{array}\right)$

To calculate f_w, then, we just need to multiply n matrices C_a of fixed degree together. By multiplying them together in pairs we can do this in $\mathcal{O}(\log n)$ parallel steps. Each of these steps potentially doubles the number of digits in the matrices' entries, and multiplying n digit numbers takes $\mathcal{O}(\log n)$ parallel time. So we get a total parallel time of

$$\mathcal{O}(\log 1 + \log 2 + \log 4 + \cdots + \log n) = \mathcal{O}(\log^2 n)$$

This is proper since (we state without proof) L_7 is in $\mathbf{NC}_1 \subset \mathbf{NC}_2$ (and in fact there are $\mathbf{NC}_1$ languages not in $\mathbf{Lin}(\mathbb{Z})$). ■

It would be nice if polynomial maps were parallelizable also, but since the composition of n polynomials of degree $k > 1$ is a polynomial of degree k^n, the space requirements grow exponentially with n.

5.4 Equation languages

Equation languages are an amusing source of examples for dynamical recognizers: for instance, the set of words in $A = \{0, 1, \times, =\}$ of the form "$w_1 \times w_2 = w_3$" where $\overline{w}_1 \overline{w}_2 = \overline{w}_3$, such as "$101 \times 11 = 1111$". We can also consider inequalities such as "$10 \times 11 > 10 + 11$". We will write $[E]_b$ for the language corresponding to an equation E expressed in base b. Then

Theorem 14. *$[E]_b$ is in $\mathbf{Lin}(\mathbb{Z})$ for any E involving $+$ and $\times$ (with $\times$ given precedence).*

Proof. We read in the first variable w_1 by letting $x_0 = 0$ and $f_n(x) = bx + n$ for $0 \leq n < b$; then $x_{w_1} = \overline{w}_1$. (This maps w_1 to the integer $\overline{w}_1$ it represents, rather than to a real in the unit interval as before.) Then we inductively proceed as follows.

If the next operation is a $+$, we store x and evaluate what is being added to it. This evaluation will conclude when we reach the next $+$ or the $=$. We then add the two together.

If the next operation is a $\times$, let a new variable be $y_0 = 0$ and use the functions $f_n(y) = by + nx$. Then $x_{w_1 \times w_2} = \overline{w}_1 \overline{w}_2$.

Finally, on reading the = (or > or whatever), simply store x, evaluate the right-hand side in the same way, and compare them. ■

This shows that $\mathbf{Lin}(\mathbb{Z})$ can be considerably more expressive than regular or context-free languages. Decimal points are easily added (exercise for the reader).

Exponentiation takes a little more work:

Theorem 15. *$[E]_b$ is in $\mathbf{Poly}_{b+1}(\mathbb{Z})$ for any E involving $+$, $\times$ and $\uparrow$ (exponentiation), in order of increasing precedence. If the only occurrences of $\uparrow$ are of the form $w_1 \uparrow w_2$ where w_1 is a constant, then $[E]_b$ is in $\mathbf{Poly}_b(\mathbb{Z})$.*

Proof. To evaluate $w_1 \uparrow w_2$, read in $x = \overline{w}_1$ as before. When you read the $\uparrow$, prepare b variables $a_n = x^n$ for $0 \leq n < b$. Let another variable be $y_0 = 1$, and let $f_n(y) = a_n y^b$ thereafter; this is a polynomial of order $b+1$, or order b if w_1 and the a_n are constants. Then $x_{w_1 \uparrow w_2} = \overline{w}_1^{\overline{w}_2}$. The rest of the evaluation can take place as before. ■

With non-determinism, we can add a sort of exponentially bounded existential quantifier. Consider equations E such as "$w_1^2 + x^2 = w_2^2$ for some $x < m$," a member of which is "$100 \uparrow 2 + x \uparrow 2 = 101 \uparrow 2$". Then we have the following:

Lemma 10. *For any integer constant c, non-deterministic linear maps can prepare a variable x with any integer value in the range $0 \leq x < c^l$ in l steps.*

Proof. Let $x_0 = 0$ and non-deterministically choose among the maps $f^{(n)}(x) = cx + n$, $0 \leq n < c$. ■

Theorem 16. *Let E be an equation with a finite number of variables x_i bound by quantifiers of the form $\exists x_i < m_i$. Let l be the total length of the input word, and let l_i and r_i be the leftmost and rightmost positions at which x_i appears. Then $[E]_b$ is in:*

1) $\mathbf{NLin}(\mathbb{Z})$ if E involves only $+$ and $\times$ and $m_i < c^{l_i}$ for some constant c

2) $\mathbf{NPoly}_{b+1}(\mathbb{Z})$ if E involves $+$, $\times$ and $\uparrow$ but not terms of the form $w \uparrow x$, and $m_i < c^{l_i}$

3) $\mathbf{NPoly}_k(\mathbb{Z})$ if E is a fixed polynomial of degree k in the w_i and x_i and $m_i < c^l$

4) $\mathbf{NPoly}_{\max(k,c+1)}(\mathbb{Z})$ if E is a fixed polynomial of degree k of terms including $w \uparrow x$ and $m_i < c^{l-r_i}$ or $m_i \propto l - r_i$ if $c = 1$

5) $\mathbf{NPoly}_{\max(k,c)}(\mathbb{Z})$ if E is a fixed polynomial of degree k of terms including $w \uparrow x$ for constant w and $m_i < c^l$ or $m_i \propto r_i$ if $c = 1$.

Proof. In cases 1 and 2, we have l_i steps of the input with which to prepare x_i with a value up to c^{l_i} as in lemma 10. Then we simply plug this value into the evaluation process of theorems 14 and 15.

In case 3, if E is a fixed polynomial P, we have all l steps of the input word to prepare the x_i and evaluate sums, products and exponents of the w_i. Then we can plug it all in to P at the end.

In cases 4 and 5, we can evaluate w^x by non-deterministically applying the maps f_n of theorem 15. If the exponent is in unary ($c = 1$) we can generate linearly growing values of x, while higher bases ($c > 1$) allow x to grow exponentially. If w is a constant (case 5), we know it in advance and we can use all l steps in the input word to increment x. In general (case 4), we only have the $l - r_i$ steps between the last occurrence of $w \uparrow x$ and the end of the word.

If x_i appears several times, we can easily check that we use the same value for it each time. In the first two cases we can prepare the value for its first instance, and stick to that thereafter. In the third, fourth and fifth cases each x_i only appears a finite number of times since the equation is fixed, and so we can use a different variable for each instance and check that they're all equal at the end. ■

As an example of the fifth case, the language of powers of 3 in binary

$$\{1, 11, 1001, 11011, 1010001, \ldots\} = [\exists x < l : w = 3^x]_2$$

is in $\mathbf{NLin}(\mathbb{Z})$. Just let $y_0 = 1$, non-deterministically multiply y by 3 or leave it alone, and check that $y = \overline{w}$ at the end. It is also in $\mathbf{PieceLin}(\mathbb{Z})$, since we can multiply y by 3 whenever $3y \leq \overline{w}$ as we read in $\overline{w}$.

The reader may also enjoy showing that $w!$ can be understood in equations by $\mathbf{Poly}_2(\mathbb{Z})$ if w is written in unary, and that the language

$$\{1, 10, 110, 11000, 111000, 1011010000, 1001110110000, \ldots\}$$

of factorials $n!$ written in binary is in $\mathbf{NPoly}_2(\mathbb{Z})$ (and in $\mathbf{PiecePoly}_2(\mathbb{Z})$ as defined below).

Two obvious generalizations of theorems 14, 15 and 16 come to mind. First, with real coefficients we can name various real constants and use them in equations (although not on the right-hand side of a $\uparrow$). Secondly, by maintaining an evaluation stack, we can parse parentheses up to a bounded number of levels.

5.5 Real coefficients

We end this section with two simple results about linear and polynomial recognizers with real, rather than integer or rational, coefficients.

Theorem 17. ***PieceLin***$(\mathbb{R})$ *and* $\mathbf{Poly}_2(\mathbb{R})$ *each contain all languages on a one-symbol alphabet.*

Proof. Consider recognizers on a one-symbol alphabet $\{a\}$ where $f_a(x) = 2x \bmod 1$ (piecewise-linear) or $4x(1-x)$ (quadratic). Both of these map the half-intervals $[0, 1/2)$ and $[1/2, 1]$ onto the entire unit interval. For any initial point x_0, we can define an *itinerary*

$$s_t = \begin{cases} 0 \text{ if } f_a^t(x_0) < 1/2 \\ 1 \text{ if } f_a^t(x_0) \geq 1/2 \end{cases}$$

showing which half of the interval x falls into as f_a is iterated. For $f_a(x) = 2x \bmod 1$ this is just x_0's binary digit sequence.

If $H_{\text{yes}} = [1/2, 1]$, then, $L_\rho = \{a^t \mid s_t = 1\}$. Both these maps have *complete symbolic dynamics* [20], i.e. there is an x_0 for every possible itinerary; so we can get any $L_\rho \subset \{a\}^*$ we want by properly choosing x_0. ■

Corollary. *The class* $\mathcal{C}(\mathbb{R})$ *properly contains* $\mathcal{C}(\mathbb{Z})$ *for* $\mathcal{C}$ = **Lin**, **NLin**, **PieceLin**, **NPieceLin**, **Poly**, **NPoly**, $\mathbf{Poly}_k$ *and* $\mathbf{NPoly}_k$ *for all* k, **Elem** *and* **NElem**.

Proof. Theorem 17 shows that $\mathcal{C}(\mathbb{R})$ is uncountable for all these classes except **Lin** and **NLin**. These are uncountable as well; for instance, for each angle ϕ there is a distinct language $L_\phi \subset a^*$ in $\mathbf{Lin}(\mathbb{R})$ recognized by an f_a that rotates the plane by ϕ and accepts whenever x_{a^n} is in the upper half-plane.

On the other hand, $\mathcal{C}(\mathbb{Z})$ is countable for all these classes, since any recognizer with integer or rational coefficients can be described with a finite list of integers. So $\mathcal{C}(\mathbb{Z})$ is of smaller cardinality than $\mathcal{C}(\mathbb{R})$. ■

6 The polynomial degree hierarchy

We will call the classes $\mathbf{Poly}_k$ and $\mathbf{NPoly}_k$ the deterministic and non-deterministic *polynomial degree hierarchies* (not to be confused with the polynomial hierarchy $\Sigma_k\mathbf{P}$ of discrete computation theory). Are these hierarchies distinct? That is, does $\mathbf{Poly}_{k+1}$ properly contain $\mathbf{Poly}_k$ for all k? Or do they collapse, so that there a k such that $\mathbf{Poly}_j = \mathbf{Poly}_k$ for all $j > k$?

Conjecture 4. *Both the deterministic and non-deterministic polynomial degree hierarchies are distinct.*

Proof? We have already shown (theorem 9) that the lowest two levels are distinct in the deterministic case. We can imagine several methods of proof for the entire hierarchy.

First, we could refine the argument of theorem 7 to produce a series of languages L_k each recognizable in $\mathbf{Poly}_k$ but out-stripping the ability of polynomials of smaller degree to produce independent sets.

Secondly, we could use polynomials of degree $k+1$ to simulate all possible polynomials of degree k by representing their constants with additional variables, and then introduce some kind of diagonalization.

Thirdly, we can connect distinctness to the idea that we can't recognize equation languages unless we actually calculate the quantities in them:

Lemma 11. *If equation languages involving terms of the form* $w_1 \uparrow w_2$ *cannot be recognized without some variables reaching values of at least* $\mathcal{O}(\overline{w}_1^{\overline{w}_2})$*, then the polynomial degree hierarchy is distinct.*

Proof. An expression of the form $w_1 \uparrow w_2$ with length l in base b can have a value of $\mathcal{O}(\overline{w}_1^{b^l})$, while polynomials of degree k can only reach $\mathcal{O}(c^{k^l})$ in l steps. If the premise is true, then, the languages $[E]_b$ of theorem 15 with constant w_1 are each in $\mathbf{Poly}_b$ but not $\mathbf{Poly}_k$ for $k < b$. ■

Fourth, distinctness is equivalent to the conjecture that, for each k, $\mathbf{Poly}_k$ and $\mathbf{NPoly}_k$ lack a particular closure property:

Lemma 12. *For any $j > k \geq 2$, any language in $\mathbf{Poly}_j$ is a non-alphabetic inverse homomorphism of a language in $\mathbf{Poly}_k$. Therefore, the (deterministic) polynomial degree hierarchy collapses to level $k \geq 2$ if and only if $\mathbf{Poly}_k$ is closed under non-alphabetic inverse homomorphism. Similarly for* **NPoly**.

Proof. Let h_n be the non-alphabetic homomorphism that repeats each symbol n times, e.g. $h_3(abca) = aaabbbcccaaa$. We will show that for any L in $\mathbf{Poly}_j$ and any $k > 2$, $h_n(L)$ is in $\mathbf{Poly}_k$ for some n.

A polynomial of degree j can be written as the composition of $n = \lceil \log_k j \rceil$ polynomials of degree k, for any $k \geq 2$. This composition can be carried out by a finite-state control with n states. For the body of the word, then, n repetitions of each symbol allow a $\mathbf{Poly}_k$-recognizer to simulate a $\mathbf{Poly}_j$-recognizer.

But for the last symbol, we need to simulate f_a and also calculate the measurement functions h, giving polynomials $h \circ f_a$ of degree j^2. This requires $\lceil \log_k j^2 \rceil$ polynomials of degree k. One of these can be provided by the new measurement functions, so $\lceil 2 \log_k j \rceil - 1$ repetitions of the last symbol suffice.

So for any L in $\mathbf{Poly}_j$ and any $k > 2$, $h_n(L)$ is in $\mathbf{Poly}_k$ where $n = \lceil 2 \log_k j \rceil - 1$. If $\mathbf{Poly}_k$ is closed under non-alphabetic inverse homomorphism, then, L is in $\mathbf{Poly}_k$ since $h_n(L)$ is and the hierarchy collapses.

Conversely, if L is in $\mathbf{Poly}_k$ and h is a non-alphabetic homomorphism that maps symbols onto words of length at most n, then $h^{-1}(L)$ is in $\mathbf{Poly}_{k^n}$ since, as in lemma 3, each step is the composition of n polynomials of degree k. So if the hierarchy collapses, $\mathbf{Poly}_k$ is closed under h^{-1}. ■

Corollary 1. *If $\mathbf{Poly}_k = \mathbf{Poly}_{k^2}$ for some $k > 1$, then $\mathbf{Poly}_j = \mathbf{Poly}_k$ for all $j > k$. Similarly for* **NPoly**.

Proof. If $\mathbf{Poly}_k = \mathbf{Poly}_{k^2}$, then $\mathbf{Poly}_k$ is closed under inverse homomorphisms that at most double the length of words. But by composing these, we can get any homomorphism we want, so $\mathbf{Poly}_k$ is closed under inverse homomorphisms in general and lemma 12 applies. ■

We can improve this to the following, analogous to standard lemmas in recursion theory:

Corollary 2. *If $\mathbf{Poly}_k = \mathbf{Poly}_{k+1}$ then $\mathbf{Poly}_j = \mathbf{Poly}_k$ for all $j > k$, and similarly for* **NPoly**.

Proof. Recall [21] that a *generalized sequential machine* (GSM) is a finite-state machine that converts an input word into an output. If L is in $\mathbf{Poly}_k$ and a GSM mapping M increases the length of words by a factor of at most m, then $M^{-1}(L)$ is in $\mathbf{Poly}_{k^m}$.

Therefore, if $\mathbf{Poly}_k = \mathbf{Poly}_{k+1}$, then $\mathbf{Poly}_k$ is closed under inverse GSM mappings that increase the length of the word by at most $m = \log_k(k+1)$. It is easy to show that we can get any homomorphism we like by composing GSM mappings with any $m > 1$, except on words of length less than $1/(m-1)$ which cannot increase in length. But this is a finite set of exceptions which we can catch with additional variables, so $\mathbf{Poly}_k$ is closed under all inverse homomorphisms and lemma 12 applies again. ■

It hardly seems possible that the composition of any number of polynomials can be simulated by a single polynomial of the same degree; but this is exactly what it would mean for some $\mathbf{Poly}_k$ to be closed under arbitrary inverse homomorphisms. Therefore, we consider lemma 12 strong evidence for distinctness.

We note that we cannot prove distinctness, even in the deterministic case, using VC-dimension: since it has an upper bound of $\mathcal{O}(nd \log k)$ [17], polynomials of degree k in $\mathbb{R}^d$ could conceivably be simulated by quadratic polynomials in $\mathbb{R}^{\mathcal{O}(d \log k)}$.

A proof of conjecture 4 seems just around the corner. We invite clever readers to complete it!

7 Higher recognizer classes

7.1 Elementary functions

We now consider the classes **Elem** and **NElem**, where we allow exponential, trigonometric and polynomial functions, as well as their compositions. In **Elem**($\mathbb{Z}$) we allow coefficients that are elementary functions of integers, such as rational or algebraic numbers.

Theorem 18. **Elem**($\mathbb{Z}$) *properly contains* **Poly**($\mathbb{Z}$).

Proof. We will show that the language L_7 of theorem 7 is in **Elem**($\mathbb{Z}$). Recall its definition:

$$L_7 = \{ w_1 \sharp w_2 \sharp \cdots \sharp w_m \natural v \mid v = w_i \text{ for some } i \}$$

By reading in $\overline{w}$ as in theorem 12, and letting $x_0 = 0$ and $f_\sharp(x) = x + 2^{\overline{w}}$, we can construct

$$x = \frac{\sum_i 2^{\overline{w}_i}}{2^{\overline{v}}} \in \left\{ \begin{array}{l} [2k+1, 2k+2) \text{ if } v = w_i \text{ for some } i \\ [2k, 2k+1) \text{ if } v \neq w_i \text{ for all } i \end{array} \right. \quad \text{for some integer } k$$

In other words, the $2^{\overline{v}}$ digit of $\sum_i 2^{\overline{w}_i}$ is 1 if $v = w_i$ and 0 otherwise. So let H_{yes} require that $\sin \pi x_w < 0$ or $\cos \pi x_w = -1$, i.e. $x \in (2k+1, 2k+2)$ or $x = 2k+1$.

Since L_7 is in **Elem**($\mathbb{Z}$) but not **Poly**($\mathbb{R}$), the inclusion **Poly**($\mathbb{Z}$) $\subset$ **Elem**($\mathbb{Z}$) is proper. ■

Here we're using the fact that all the sets $S_j = \{x \mid \sin 2^j x < 0\}$ for $j = 0, 1, 2, \ldots$ are independent, i.e. the family $\{S_j\}$ has infinite VC-dimension.

Conjecture 5. **NElem**($\mathbb{Z}$) *properly contains* **NPoly**($\mathbb{Z}$).

Proof? Consider the numbers $M_n = 2^{2^n} + 1$. Since $M_0 = M_1 - 2$ and $M_n(M_n - 2) = M_{n+1} - 2$,

$$\prod_{i=0}^{n} M_i = M_{n+1} - 2$$

so the M_n are mutually prime for all $n \geq 0$. Therefore, if $x = \prod_n M_n^{c_n}$ the c_n are unique, and we have random access to an arbitrary number of counters c_n.

For instance, consider the language of *block anagrams*

$$L_{\text{anag}} = \{w_1 \sharp w_2 \sharp \cdots \sharp w_m \natural v_1 \sharp v_2 \sharp \cdots \sharp v_m \,|\, \text{for some permutation } \pi,\ v_i = w_{\pi(i)} \text{ for all } i\}$$

By reading in $\overline{w}$, letting $p_0 = 1$ and $f_\sharp(p) = M_{\overline{w}} p$, and similarly for $\overline{v}$ and q, construct $p = \prod_i M_{\overline{w}_i}$ and $q = \prod_i M_{\overline{v}_i}$. Then let H_{yes} require that $p = q$.

Here we're accessing c_n in $\mathcal{O}(\log n)$ time for arbitrary n, and we conjecture that **NPoly**-recognizers can't do this. However, they can if we name n in unary, since $M_{n+1} = (M_n - 1)^2 + 1$ is a quadratic function of M_n. For instance, L_{anag} is in $\mathbf{Poly}_2(\mathbb{Z})$ if the w_i and v_i are over a one-symbol alphabet.

Unfortunately, besides the rather generous upper bounds given in theorems 11 and 12, we have no idea how to prove a language is outside **NPoly**, or even **NLin**.

Finally, we note that allowing arbitrary reals makes **Elem** trivial:

Theorem 19. $\mathbf{Elem}(\mathbb{R})$ *contains all languages.*

Proof. For any language L, let $x_L = \sum_{w \in L} 3^{-\overline{w}}$ (we use base 3 to avoid ambiguities in the digit sequence). Then the $3^{-\overline{w}}$ digit of x_L is 1 if $x \in L$ and 0 otherwise, so let H_{yes} require that

$$0 \leq \sin \frac{2\pi 3^{\overline{w}} x_L}{3} \leq -\sqrt{3} \cos \frac{2\pi 3^{\overline{w}} x_L}{3}$$

i.e. $\frac{2\pi 3^{\overline{w}} x_L}{3} \bmod 2\pi \in [\frac{2\pi}{3}, \pi]$ or $3^{\overline{w}} x_L \bmod 3 \in [1, 3/2]$. ■

7.2 Analytic and continuous functions

The class **Analytic** (which we will not abbreviate) is also trivial, unless we restrict ourselves to a countable set of closed forms:

Theorem 20. *The class* **Analytic** *contains all languages.*

Proof. Simply map input words to an integer $\overline{w}$, choose an analytic function h such that $h(\overline{w}) = 1$ if $w \in L$ and 0 otherwise, and require that $h(\overline{w}) \geq 1/2$. (We can also do this with piecewise-linear maps if we allow a countably infinite number of components.) ■

8 Complexity and decidability properties

Given a description of a dynamical recognizer ρ, we can ask whether $L_\rho = \emptyset$. Given ρ and an input word w, we can ask whether $w \in L_\rho$. We will refer to these problems as *emptiness* and *membership* respectively; we will show that even for the simplest classes, they are undecidable or intractable. For definitions of **P**- and **NP**-completeness, see [15].

Theorem 21. *Emptiness is undecidable for* $\mathbf{Lin}(\mathbb{Z})$ *if* $d \geq 2$*, and for* $\mathbf{Elem}(\mathbb{Z})$ *for all* d*.*

Proof. Post's Correspondence Problem (PCP) is the following: given a list of words w_i and u_i, is there a sequence $i_1, i_2, \ldots, i_k$ such that

$$w_{i_1} w_{i_2} \cdots w_{i_k} = u_{i_1} u_{i_2} \cdots u_{i_k}?$$

To reduce PCP to the non-emptiness of a $\mathbf{Lin}(\mathbb{Z})$ language, let $x_0 = y_0 = 0$, let

$$f_i(x, y) = (\mathrm{push}_{w_i}(x),\ \mathrm{push}_{v_i}(y))$$

and require that $x = y > 0$ to accept. Post's Correspondence Problem is undecidable [21].

For $\mathbf{Elem}(\mathbb{Z})$, we recall [32] that elementary functions in one dimension can simulate Turing machines with an exponential slowdown. ■

Corollary 1. *Membership is* $\mathbf{NP}$*-complete for* $\mathbf{NLin}(\mathbb{Z})$ *if* $d \geq 2$*, even for languages on a one-symbol alphabet.*

Proof. Post's Correspondence Problem is **NP-complete** [15] if we place a bound on k. Let a single map $f_a^{(i)}$ non-deterministically choose between the f_i above, or do nothing. Then ask if a^k is in L_ρ. ■

Corollary 2. *For languages in* $\mathbf{Lin}(\mathbb{Z})$*, it is undecidable whether* $L_1 \cap L_2 = \emptyset$*,* $L_1 \subseteq L_2$ *(inclusion),* $L_1 = L_2$ *(equivalence), or* $L = A^*$ *(universality).*

Proof. Emptiness is a special case of each of these, since $\mathbf{Lin}(\mathbb{Z})$ is closed under intersection, union, and complement (e.g. $L_1 \subseteq L_2$ if and only if $L_1 \cap \overline{L_2} = \emptyset$). ■

Corollary 3. *For languages in* $\mathbf{Lin}(\mathbb{Z})$*, it is undecidable whether* L *is regular, context-free,* **DCF***,* **QA***,* **NQA***, etc.*

Proof. This follows from Greibach's Theorem [19, 21], which states that virtually any non-trivial property is undecidable for a class which is closed under concatenation with a regular language (concatenation of languages with disjoint alphabets suffices, which we have by lemma 7) and union, and for which $L = A^*$ is undecidable. ■

Theorem 22. *Membership is* $\mathbf{P}$*-complete for* $\mathbf{PieceLin}(\mathbb{Z})$ *if* $d \geq 3$*, for* $\mathbf{PieceLin}(\mathbb{Q})$ *if* $d \geq 2$*, and* $\mathbf{Elem}(\mathbb{Z})$ *if* $d \geq 2$*.*

Proof. This follows from the fact that two-dimensional piecewise-linear maps with rational coefficients can simulate Turing machines in real time [30, 10]. This reduces any problem in $\mathbf{P}$ that takes time t on input w to the membership of a^t where f_a iterates the map and $x_0 = x_w$. Doing this with integer coefficients as in theorem 3 requires one more variable. Elementary functions in two dimensions can also simulate Turing machines in real time [25]. ■

Several questions suggest themselves. Is emptiness decidable for $\mathbf{Lin}(\mathbb{Z})$ if $d = 1$? Is membership still $\mathbf{P}$-complete for $\mathbf{PieceLin}(\mathbb{Z})$ if $d \leq 2$, or for $\mathbf{PieceLin}(\mathbb{Q})$ if $d = 1$? Is membership $\mathbf{P}$-hard for $\mathbf{Poly}_k(\mathbb{Z})$ for some k? Theorem 13 makes it highly unlikely that membership in $\mathbf{Lin}(\mathbb{Z})$ is $\mathbf{P}$-complete, since then we would have $\mathbf{NC}_2 = \mathbf{P}$.

9 Relationships with other models of analog computation.

There are several differences between Blum, Shub and Smale's (BSS) analog machines [3], Siegelmann and Sontag's (SS) neural networks [38], and dynamical recognizers.

First, BSS-machines can branch on polynomial inequalities during the course of the computation. Except for **PieceLin**, our recognizers have completely continuous dynamics except for the final measurement of H_{yes}. SS-machines are defined with piecewise-linear maps.

Secondly, BSS- and SS-machines are not restricted to real time, so that time complexity classes such as **P**, **EXPTIME** and so on can be defined for them.

Thirdly, BSS-machines can recognize "languages" whose symbols are real numbers, and can make real number guesses in their non-deterministic versions.

Finally, BSS-machines have unbounded dimensionality, and receive their entire input as part of their initial state. Therefore, they have at least n variables on input of length n. SS-machines, like ours, have bounded dimensionality, and receive their input dynamically rather than as part of the initial state.

This last point seems entirely analogous to Turing machines. If we wish to consider sub-linear space bounds such as **LOGSPACE**, we need to use an Turing machine which receives its input on a read-only tape separate from its worktape.

This suggests a unification of all three models. First of all, let **PiecePoly** and **NPiecePoly** be recognizer classes where the f_a are piecewise polynomials, with polynomial component boundaries (these could serve as models of "hybrid systems").

Secondly, relax our real-time restriction by iterating an additional map f_{comp}, in the same class as the f_a, until x falls into some subset H_{halt}.

Thirdly, restrict BSS-machines to their Boolean part BP and to *digital non-determinism*, e.g. $\mathbf{DNP}_{\mathbb{R}}$ [11].

And finally, define an *on-line BSS-machine* as one who receives its input dynamically in the first n steps, and which has a bound $\mathbf{SPACE}(f(n))$ on the number of variables it can use during the computation. (In [18] these are called *separated input and output* or SIO-BSS-machines.)

Then we can look at these classes in a unified way:

$$
\begin{aligned}
\mathbf{PiecePoly}(\mathbb{R})\mathbf{TIME}(\mathcal{O}(n^k))\mathbf{SPACE}(\mathcal{O}(n^k)) &= \text{BP}(\mathbf{P}_{\mathbb{R}}) \text{ (Blum, Shub and Smale [3])} \\
\mathbf{NPiecePoly}(\mathbb{R})\mathbf{TIME}(\mathcal{O}(n^k))\mathbf{SPACE}(\mathcal{O}(n^k)) &= \text{BP}(\mathbf{NDP}_{\mathbb{R}}) \text{ (Cucker and Matamala [11])} \\
\mathbf{PieceLin}(\mathbb{R})\mathbf{TIME}(\mathcal{O}(n^k))\mathbf{SPACE}(\mathcal{O}(n^k)) &= \text{BP}(\mathbf{P}^{<}_{\text{lin}}) \text{ (Meer [28] and Koiran [23])} \\
\mathbf{PieceLin}(\mathbb{R})\mathbf{TIME}(\mathcal{O}(n^k))\mathbf{SPACE}(\mathcal{O}(1)) &= \mathbf{NET-P} \text{ (Siegelmann and Sontag [38])} \\
\mathbf{PiecePoly}_k(\mathbb{Z})\mathbf{TIME}(n)\mathbf{SPACE}(\mathcal{O}(1)) &= \mathbf{PiecePoly}_k(\mathbb{Z}) \text{ (dynamical recognizers)}
\end{aligned}
$$

and similarly for other complexity classes. (The last two lines of this table contrast with the discrete case, since Turing machines with constant space can only recognize regular languages.)

Then there are a number of things we have already shown, or which are obvious:

Corollary to Theorem 7. $\mathbf{PiecePoly}$ *and* $\mathbf{PiecePoly}_k$ *for all* k *are properly contained in their non-deterministic counterparts in real time, and are not closed under reversal, CYCLE, alphabetic homomorphism or concatenation.*

Proof. The VC-dimension of $\mathbf{PiecePoly}_k$ maps in $\mathbb{R}^d$ with j components each is $\mathcal{O}(nd \log jk)$ [17]. So L_7 isn't in $\mathbf{PiecePoly}$, but the various modified versions of L_7 in corollaries 1, 3 and 4 of theorem 7 are in $\mathbf{PiecePoly}_1 = \mathbf{PieceLin}$. ■

Then, just as for deterministic Turing machines [37], linear time is more powerful than real time:

Theorem 23. *Real time* $\mathbf{TIME}(n)$ *is properly contained in linear time* $\mathbf{TIME}(\mathcal{O}(n))$ *for* $\mathbf{PiecePoly}$ *and* $\mathbf{PiecePoly}_k$ *for all* k.

Proof. We will show that $\mathbf{TIME}(\mathcal{O}(n))$ is closed under reversal for all these classes. Simply store the input with $\text{push}^{\text{right}}$ into a variable $p = \overline{w^R}$, and then use piecewise-linear maps to extract the digits in reverse order. ■

We can also conjecture, as we did for $\mathbf{Poly}$ and $\mathbf{NPoly}$:

Conjecture 6. *The deterministic and non-deterministic piecewise-polynomial degree hierarchy is distinct.*

This could only be true in real time, since quadratic maps can simulate polynomials of any degree with a constant slowdown. We also conjecture that branching is of fundamental importance, even when additional computation time is allowed:

Conjecture 7. *For all* k *and all* $f(n)$, $\mathbf{Poly}_k\mathbf{TIME}(f(n))$ *is properly contained in* $\mathbf{PiecePoly}_k\mathbf{TIME}(f(n))$, *and similarly for non-deterministic classes.*

We have already shown this for $k = 1$ in theorem 9: L_{unary} is not in $\mathbf{Lin}$ no matter how much time is allowed. Unless the degree hierarchies are distinct, VC-dimension arguments can't separate $\mathbf{Poly}_k$ from $\mathbf{PiecePoly}_k$. Since it has an upper bound of $\mathcal{O}(nd \log jk)$ where j is the number of components of each map [17], branching could conceivably be simulated by polynomials of degree jk.

Finally, we note that combining the above with results of Cucker and Grigoriev [12] and Koiran [23, 24] give us bounds on $\mathbf{Poly}$ and $\mathbf{NPoly}$ tighter than theorem 10, as well as bounds on $\mathbf{PieceLin}(\mathbb{R})$ and $\mathbf{NPieceLin}(\mathbb{R})$. Recall [34] that a machine has *polynomial advice* if it has access to an oracle whose advice is polynomially long and depends only on the length of the input, rather than on the input itself. Classes with polynomial advice are written $\mathbf{P}/poly$, $\mathbf{NP}/poly$, etc. Then:

Theorem 24. *The following containments hold:*

$$\mathbf{PiecePoly}(\mathbb{Z}) \subset \mathbf{NPiecePoly}(\mathbb{Z}) \subset \mathbf{NPiecePoly}(\mathbb{R}) \subset \mathbf{PSPACE}/poly$$
$$\mathbf{PieceLin}(\mathbb{R}) \subset \mathbf{P}/poly$$

$\mathbf{NPieceLin}(\mathbb{R}) \subseteq \mathbf{NP}/poly$

where $\subset$ *indicates proper inclusion.*

10 Conclusion and directions for further work

In addition to the conjectures and open problems mentioned above, there are several directions in which one could extend this work.

1) We have seen that **PieceLin** and **NLin** are fundamentally more powerful than **Lin**. Is **PieceLin** contained in any continuous class, and is **NLin** contained in any deterministic class? In other words, can branching and non-determinism be compensated for in real time by going to a more powerful class of functions such as **Elem**? We conjecture that these are fundamentally different computational resources, and that **NLin**, **PieceLin** and **Elem** are all incomparable.

(Two comments: **NLin** and **PieceLin** can be trivially simulated by arbitrary continuous functions, but we want smooth, closed-form functions. We also note that without the restriction of real time, $\mathbf{NP}_{\mathbb{R}} \subset \mathbf{EXPTIME}_{\mathbb{R}}$ in the BSS model [3].)

2) As alluded to in lemma 11, let **DIGITS** be the maximum number of digits in a recognizer's variables as a function of the input length n. This is a computational resource, analogous to space in Turing machines, and (for variables in $\mathbb{Z}$) proportional to the logarithm of the volume in $\mathbb{R}^d$ the recognizer needs.

DIGITS is also related to the robustness of a dynamical recognizer with respect to noise. If our variables are rational and confined to the unit cube, and the system is exposed to noise of size $\epsilon = \mathcal{O}(2^{-d})$, then words in a language in $\mathbf{DIGITS}(f(n))$ will be correctly recognized up to length $n = f^{-1}(\log \epsilon^{-1})$. Mike Casey has shown that, in the presence of noise, finite-dimensional dynamical recognizers can only recognize arbitrarily long words for regular languages [8].

For **Lin** and $\mathbf{Poly}_k$, **DIGITS** is limited to $\mathcal{O}(n)$ and $\mathcal{O}(k^n)$ respectively. Beneath these bounds, or for **Elem**, are hierarchies based on **DIGITS** distinct? For instance, is $\mathbf{DIGITS}(f(n))$ properly contained in $\mathbf{DIGITS}(g(n))$ if $\lim_{n\to\infty} f(n)/g(n) = 0$? **DIGITS** is at least linear for any language such as $L_{\rm pal}$ or $L_{\rm copy}$ where every word u has a unique set of words v such that $uv \in L$, since the recognizer has to represent all 2^n possible u's in a unique way. But besides this trivial observation, how can we prove lower limits on **DIGITS**?

3) As a purely automata-theoretic question, it would be nice to show that obstinate PDAs and QAs are strictly less powerful than their deterministic counterparts (**OCF** is a subset of the *input-driven* CFLs), and that CQAs are strictly less powerful than QAs in the obstinate, deterministic, and non-deterministic cases.

4) As generalizations of non-determinism, we could consider alternating real time, or probabilistic models analogous to **ZPP**, **BPP** or **RP** [34].

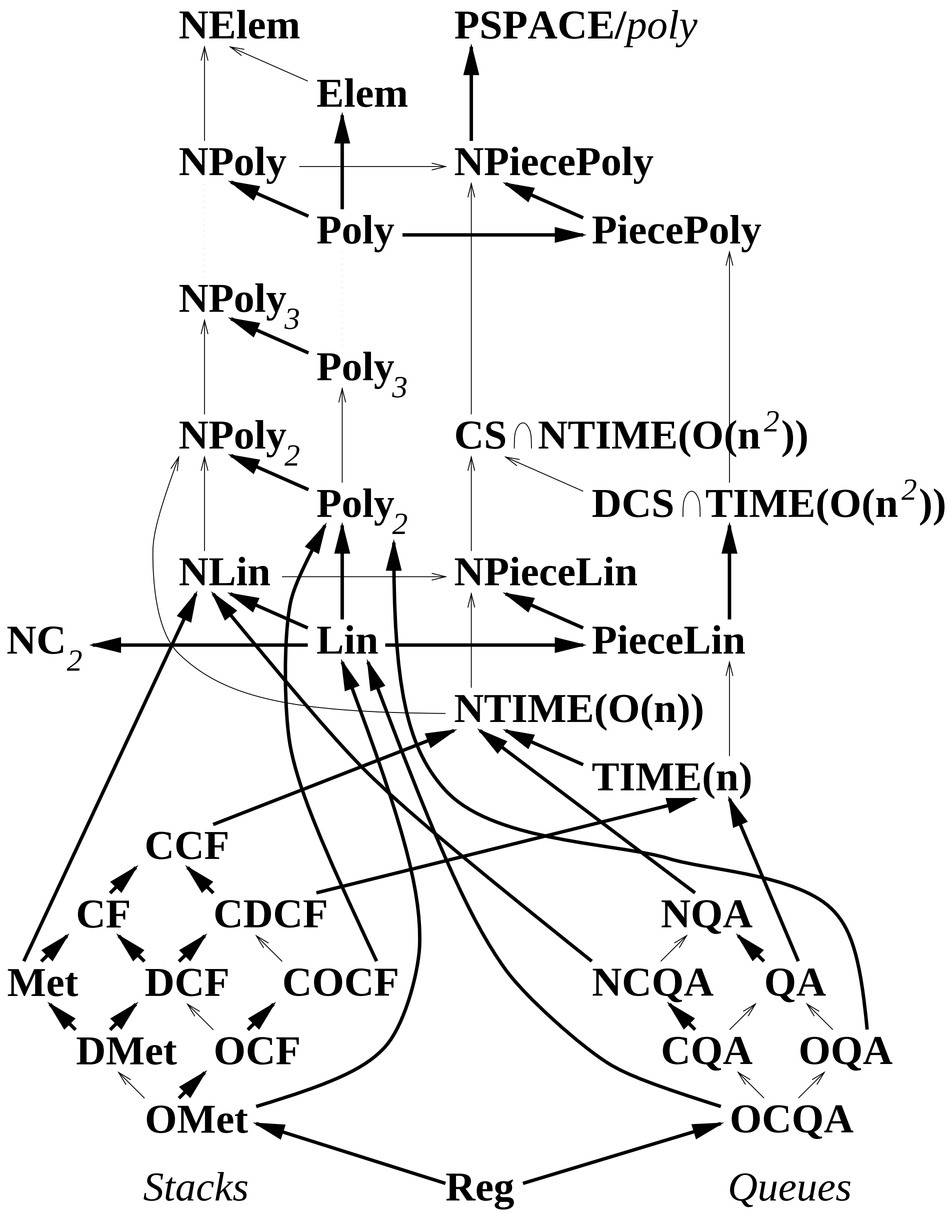

Figure 3: A summary of the inclusions proved in this paper. Inclusions known to be proper are in bold.

5) In several cases (theorem 9 vs. theorem 18, and theorem 17 vs. theorem 19) quadratic maps seem to be roughly equivalent to elementary maps when their input is given in unary. How deep does this equivalence go?

6) Can we exhibit a language not in $\mathbf{Elem}(\mathbb{Z})$? Can we exhibit a language not in $\mathbf{NLin}(\mathbb{Z})$, other than by using the complexity bounds in theorem 12? Can we exhibit a language not in $\mathbf{NLin}(\mathbb{R})$? We need methods other than VC-dimension for these.

7) Is it possible to define natural reductions or transducers for these classes, and if so, do they have natural complete problems? (I thank the referee for suggesting this question.)

8) What relation, if any, do our classes **Lin**, **Poly** and **Elem** have to function-valued complexity measures such as those in [5] and [35]? The VC-dimension argument, for instance, can be seen as a refinement of the latter; they count the number of equivalence classes where two words are equivalent if they can be followed by the same suffixes, while the VC-dimension measures how independently sets of allowed suffixes overlap.

9) Finally, we believe that sub-linear space classes in these models (such as $\mathbf{PiecePoly}(\mathbb{Z})\mathbf{TIME}(\mathcal{O}(n))\mathbf{SPACE}(\mathcal{O}(\log n))$, linear time and logarithmic space) are very much worth studying. It ought to be possible to prove space hierarchy theorems within each time class analogous to those for Turing machines (constant space is universal if unlimited time is allowed [29]). Grädel and Meer have given a logical description of the polynomial time, constant space class [18].

In figure 3 we summarize the inclusions between language classes, both dynamical and discrete, that we have been able to prove or which we already knew.

Acknowledgements. I thank Elizabeth Hunke and Mats Nordahl for careful readings of the manuscript; Jean-Camille Berget, Mike Casey, Kiran Chilakamarri, Patrick Dymond, Jeff Erickson, Michael Fischer, Christian Herzog, Volker Heun, Martin Huehne, Tao Jiang, Georg Karner, Ilias Kastanas, Marco Ladermann, Torben Mogensen, Ian Parberry, Jordan Pollack, Vicki Powers, Danny Raz, Kenneth Regan, Hava Siegelmann, Janos Simon, D. Sivakumar and Burkhard Stubert for helpful communications; and Spootie the Cat for companionship.

References

[1] R. Bartlett and M. Garzon, "Computational complexity of piecewise linear maps of the interval." Submitted to *Theoretical Computer Science*.

[2] J. Berstel, *Transductions and Context-Free Languages.* Teubner Studienbücher, 1978.

[3] 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** (1989) 1–46.

[4] A. Blumer, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth, "Learning and the Vapnik-Chervonenkis dimension." *Journal of the ACM* **36(4)** (1989) 929–965.

[5] L. Boasson, B. Courcelle and M. Nivat, "The rational index: a complexity measure for languages." *SIAM J. Comput.* **10(2)** (1981) 284–296.

[6] R. Book and S. Greibach, "Quasi-realtime languages." *Math. Systems Theory* **4** (1970) 97–111.

[7] F.J. Brandenburg, "Intersections of some families of languages." In *Proc. of 13th ICALP,* Lecture Notes in Computer Science, Vol. 226 61–68. Springer-Verlag, 1986.

[8] M. Casey, "The dynamics of discrete-time computation, with application to recurrent neural networks and finite-state machine extraction." *Neural Computation* **8:6** (1996)

[9] A. Cherubini, C. Citrini, S. Crespi Reghizzi and D. Mandrioli, "QRT FIFO automata, breadth-first grammars and their relations." *Theoretical Computer Science* **85** (1991) 171–203.

[10] M. Cosnard, M. Garzon, and P. Koiran, "Computability properties of low-dimensional dynamical systems." *Theoretical Computer Science* **132** (1994) 113–128.

[11] F. Cucker and M. Matamala, "On digital nondeterminism." To appear in *Math. Systems Theory.*

[12] F. Cucker and D. Grigoriev, "On the power of real Turing machines over binary inputs." NeuroCOLT Technical Report NC-TR-94-007 (1994).

[13] S. Das, C.L.Giles, and G.Z.Sun, "Using prior knowledge in an NNPDA to learn languages." *Advances in Neural Information Processing Systems* **5** (1993) 65–72.

[14] J. Elman, "Language as a dynamical system." In R.F. Port and T. van Gelder (Eds.), *Mind as Motion: Explorations in the Dynamics of Cognition.* MIT Press, 1995.

[15] M.R. Garey and D.S. Johnson, *Computers and Intractibility: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

[16] C.L. Giles, C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun, and Y.C. Lee, "Learning and extracting finite-state automata with second-order recurrent networks." *Neural Computation* **2** (1992) 331–349

[17] P.W. Goldberg and M.R. Jerrum, "Bounding the Vapnik-Chevonenkis dimension of concept classes parametrized by real numbers." *Machine Learning* **18** (1995) 131–148.

[18] E. Grädel and K. Meer, "Descriptive complexity theory over the real numbers." NeuroCOLT Technical Report NC-TR-95-040 (1995), and in *Proc. of the 27th Symposium on the Theory of Computing* (1995) 315–324.

[19] S.A. Greibach, "A note on undecidable properties of formal languages." *Math. Systems Theory* **2: 1** (1968) 1–6.

[20] J. Guckenheimer and P. Holmes, *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields.* Springer-Verlag, 1983.

[21] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[22] D.E. Knuth, *Seminumerical Algorithms.* Addison-Wesley, 1981.

[23] P. Koiran, "Computing over the reals with addition and order." *Theoretical Computer Science* **133** (1994) 35–47.

[24] P. Koiran, "A weak version of the Blum, Shub and Smale model." DIMACS Technical Report 94–10 (1994).

[25] P. Koiran and C. Moore, "Closed-form analytic maps in one and two dimensions can simulate universal turing machines." To appear in *Theoretical Computer Science.*

[26] B.L. Leong and J.I. Seiferas, "New real-time simulations of multihead tape units." *Journal of the ACM* **28(1)** (1981) 166–180.

[27] L.Liu and P.Weiner, "An infinite hierarchy of intersections of context-free languages." *Math. Systems Theory* **7** (1973) 185–192.

[28] K. Meer, "Real Number Models under Various Sets of Operations." *Journal of Complexity* **9** (1993) 366–372.

[29] C. Michaux, *Differential fields, machines over the real numbers and automata.* Ph.D. thesis, Université de Mons Hainaut, Faculte des Sciences (1991).

[30] C. Moore, "Unpredictability and undecidability in dynamical systems." *Physical Review Letters* **64** (1990) 2354–2357, and *Nonlinearity* **4** (1991) 199–230.

[31] C. Moore, "Generalized one-sided shifts and maps of the interval." *Nonlinearity* **4** (1991) 727–745.

[32] C. Moore, "Smooth one-dimensional maps of the interval and the real line capable of universal computation." Santa Fe Institute Working Paper 93-01-001.

[33] U. Nilsson and F. Hald, *Her er en lille gris.* Gyldendal, 1994.

[34] C.H. Papadimitriou, *Computational Complexity.* Addison-Wesley, 1994.

[35] J. Paradaens and R. Vincke, "A class of measures on formal languages." *Acta Informatica* **9** (1977) 73–86.

[36] J. Pollack, "The Induction of Dynamical Recognizers." *Machine Learning* **7** (1991) 227–252.

[37] A.L. Rosenberg, "Real-time definable languages." *Journal of the ACM* **14** (1967) 645–662.

[38] H. Siegelmann and E.D. Sontag, "Analog Computation Via Neural Networks." *Theoretical Computer Science* **131** (1994) 331–360.

[39] M. Steijvers and P.D.G. Grünwald, "A recurrent network that performs a context-sensitive prediction task." NeuroCOLT Technical Report NC-TR-96-035 (1996), and in *Proceedings of the 18th Annual Conference of the Cognitive Science Society.* Erlbaum (in press).

[40] H.E. Warren, "Lower bounds for approximation by nonlinear manifolds." *Trans. Amer. Math. Soc.* **133** (1968) 167–178.

[41] J. Wiles and J. Elman, "Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent networks." *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society.* MIT Press, 1995.