Primitive Recursion Without Composition

Dynamical Characterizations, from Neural Networks to Polynomial ODEs

Olivier Bournez
Institut Polytechnique de Paris, Ecole Polytechnique, LIX, Palaiseau, France

Abstract

What computational mechanisms do recurrent neural networks, polynomial ordinary differential equations, and discrete polynomial maps each bring to the table, and what do they lack? All three are models of computation over the continuum: they operate on real-valued states and evolve by real-valued dynamics, even when the functions we ask them to compute are ultimately discrete. We investigate how these models compare, their strengths, their limitations, and the precise resources on which each one relies, through the lens of primitive recursive functions.

We prove that the classical notion of primitive recursion admits equivalent characterizations in all three dynamical frameworks: bounded iteration of a fixed recurrent ReLU network, robust computation by a fixed polynomial ordinary differential equation, and iteration of a fixed polynomial map in discrete time with an externally supplied step-size parameter. In each case, the time bound is itself primitive recursive, composition is not postulated as a closure rule but emerges from the dynamics, and the input is given as a raw integer vector with no auxiliary encoding. At the proof level, every primitive recursive function is first compiled into bounded iteration of a single threshold-affine normal form map, which is then interpreted as a recurrent ReLU computation on the one hand, and as a robust polynomial ODE on the other.

The equivalences expose a structural asymmetry between discrete and continuous polynomial computation. We prove that no fixed polynomial map can round uniformly toward the nearest integer, and that none can realize exact phase selection: two operations that polynomial ODEs perform robustly through their continuous-time flow. Each formalism compensates for a limitation that the others do not share: the ReLU gate provides exact branching, continuous time provides autonomous rounding and control, and the step-size parameter recovers both at the cost of discretization precision. Our equivalence theorem characterizes what each resource contributes, and opens the way to dynamical characterizations of subrecursive hierarchies and complexity classes by restricting the time bounds, polynomial degrees, or discretization resources within the same framework.

More broadly, the constructions reveal that these real-valued models do not compute by composing subroutines in the classical sense: they compute by shaping the trajectory of a dynamical system, through clocks, phase selectors, stabilization mechanisms, and error correction built into the dynamics itself. This is a mode of computation that differs structurally from symbolic programming, and our equivalence theorem provides a precise framework in which the difference can be studied.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Computability; Theory of computation → Complexity classes; Mathematics of computing → Ordinary differential equations; Computer systems organization → Analog computers; Computer systems organization → Neural networks

Keywords and phrases Discrete ordinary differential equations, Finite Differences, Implicit complexity, Recursion scheme, Ordinary differential equations, Models of computation, Analog Computations, Formal neural networks

Digital Object Identifier 10.4230/LIPIcs.MFCS.2026.74

Funding *Olivier Bournez*: ANR Project OTC.

1 Introduction

Over the integers, the theory of computation rests on a remarkably stable foundation: Turing machines, register machines, the λ-calculus, and many other formalisms are all known to define the same notion of effective computation. Over the reals, the situation is less settled. Several models have been proposed with different motivations, including computable analysis [40] and algebraic models [7], but they are provably inequivalent [40, 16]. Within this landscape, two families of models have recently emerged as *genuinely computational* rather than merely descriptive, and both have developed substantial theory.

The first comes from neural computation. Already in the early 1990s, recurrent networks with threshold-like or analog activations were shown to possess significant computational power and nontrivial complexity-theoretic structure [37, 4, 29, 36]. More recent architectures, most notably transformer models, rely on substantially different mechanisms, and only make these foundational questions more pressing. What matters here is not just that such models are expressive, but that they are *programmable*, in a way quite unlike classical symbolic programming.

The second family comes from the analog-computing (General Purpose Analog Computer [34]) tradition and, more broadly, from polynomial Ordinary Differential Equations (ODEs). A long line of work has established polynomial ODEs as a robust and machine-independent framework for computability and complexity over the reals [31, 15, 8, 14, 12, 11, 6]. Again, the point is not merely that such systems can simulate classical models, but that computations can be synthesized through mechanisms natural to continuous dynamics: oscillation, stabilization, transfer of intermediate quantities, and correction of errors along the flow.

Despite their apparent differences, these two families share a common feature: they realize computation by shaping the evolution of a dynamical system. This is a very different constructive logic from the classical one based on explicit symbolic composition. In recurrent networks and in polynomial ODEs, one often obtains a computation not by writing down its decomposition into subroutines, but by designing a state space, a feedback mechanism, a thresholding behavior, or a flow with the right dynamical properties.

A first indication that this viewpoint is genuinely different is that it changes what is easy and what is hard. Rounding to the nearest integer, for instance, is almost free in continuous time (a polynomial ODE built on $y'' = -y$ contracts to the lattice) but is provably costly or impossible for a pure polynomial map in discrete time. Section 2 makes this asymmetry precise; it is the dynamical fact that organizes the rest of the paper.

This leads to a natural foundational question. Instead of taking machines or syntactic closure principles as the primary point of departure, can one find a simple classical class of functions that already sits at the intersection of these dynamical modes of computation? Through this paper, we argue that the right place to begin is to get back to the historical roots of computability theory: the old but remarkably robust class of primitive recursive functions. We assume the reader has some familiarity with computability theory, particularly with primitive recursive functions: [32] provides a thorough reference.

Primitive recursion as a dynamical system. The class $\mathcal{PR}$ of primitive recursive functions is usually defined as the smallest class of functions over the natural numbers containing the zero, successor, and projection functions, and closed under composition and primitive recursion. Many subrecursive and complexity-theoretic classes are obtained by restricting these schemes, including the Grzegorczyk hierarchy [26, 25], polynomial-time functions [21],

and polynomial-space functions [39]; see [20] for a broad overview. In all these definitions, closure under composition is taken for granted: this is somehow the one rule that no formalism considers optional.

Primitive recursion already admits a dynamical reading. The scheme

$$f(0,\mathbf{x}) = g(\mathbf{x}), \qquad f(n+1,\mathbf{x}) = h(n, f(n,\mathbf{x}),\mathbf{x})$$

can be viewed as the bounded iteration of the transition $(n, a, \mathbf{x}) \mapsto (n+1, h(n, a, \mathbf{x}), \mathbf{x})$, where a plays the role of an accumulator: in this view, the computation is not assembled from composed subroutines, but unfolds as a single trajectory. This is reminiscent of how polynomial ODEs compute in the analog tradition, where a single continuous flow replaces any composition rule, and the required control emerges from the dynamics itself through basins of attraction, stabilization, and autonomous switching mechanisms.

The same pattern appears in recurrent neural networks, though this is not always made explicit. A recurrent network with activation ρ repeatedly applies a fixed update rule to a state vector, where one step is a finite composition of affine maps and coordinatewise applications of ρ. The computation is the trajectory $z_{n+1} = R(z_n)$ of a fixed dynamical system. When ρ is $\mathrm{ReLU}(t) = \max(t, 0)$, the system is piecewise linear. When ρ has bounded range, as for the sigmoid activations used in practice, the dynamics is confined to a bounded region of state space.

This leads to a concrete version of our foundational question: can primitive recursive functions be characterized as the functions computable by bounded iteration of such a recurrent network, with no explicit closure under composition as a basic external assumption, but composition emerging from the dynamics.

Main results. The answer is yes, and the characterization extends to all the dynamical frameworks discussed above. Our main theorem states:

▶ **Theorem 1** (Main theorem, informal[1]). *For a function $f : \mathbb{N}^d \to \mathbb{N}^e$, the following are equivalent:*

1. *f is primitive recursive in the classical sense;*
2. *f is computable by bounded iteration of a fixed recurrent ReLU network;*
3. *f is computable by bounded iteration of a fixed recurrent ρ-activation network with bounded precision, over a compact domain;*
4. *f is computable robustly by a fixed polynomial ODE;*
5. *f is computable by iteration of a fixed polynomial map in discrete time, provided a sufficiently small step size is supplied as an external parameter.*

In each case, "bounded" means that the relevant quantity, e.g. the number of iterations, the observation time, the domain diameter, or the precision, is itself primitive recursive as a function of the input. The activation ρ is only required to be primitive recursively computable in a natural sense, a condition that is satisfied by all sigmoid functions considered in practice.

In Items (2) and (4)–(5), a crucial point is that the computation starts from the raw input state $(x, 0, \ldots, 0)$: the integer input x is embedded directly in $\mathbb{R}^m$, with no auxiliary encoding. Item (3) necessarily departs from this convention, since a bounded domain cannot accommodate arbitrarily large integers as raw values; there, the input is supplied as a rational number under a fixed, standard encoding, but, in the same spirit, no problem-specific preprocessing is involved. Namely, we consider $\nu(n) = 2^{-n}$ to encode integer n, i.e we encode n as a dyadic, but this choice is arbitrary.

[1] Formal version is Theorem 20.

▶ Remark 2. Our raw-input assumption is essential, not merely a stylistic choice: it reveals the strength of the statement. Without it, the results would be considerably easier to prove but would say far less: the computational work could be hidden in the encoding step rather than exposed in the dynamics. As stated, our theorems guarantee that the entire computation is carried out by the dynamics of the system itself, with no external help.

The equivalence (1) ⇔ (2) says that composition is redundant: bounded iteration of a single ReLU network already generates the full class. The equivalence (2) ⇔ (3) shows that any well-behaved activation ρ yields the same computational power, provided domain and precision are controlled. The passage (2) ⇒ (4) eliminates the threshold gates by moving to continuous time, where a polynomial ODE autonomously generates the required control through mechanisms like the one described above. The passage (4) ⇒ (5) discretizes the ODE, but the control that was free in continuous time now reappears as the step-size parameter: a genuine computational resource. The converse (5) ⇒ (2) states that once the map and the time bound are fixed, the induced evolution on rational states is primitive recursive.

What these characterizations reveal. The five formulations are equivalent, but they make visible genuinely different *costs of computation*. In the ReLU model (2), the implicit threshold gate provides exact switching for free, but this is a strong assumption on the activation. The ρ-activation model (3) shows that any reasonable nonlinearity suffices, at the price of controlling domain and precision. In the polynomial ODE (4), no discrete activation is needed at all: the dynamics itself generates rounding, switching, and error correction, but requires continuous time. In the discrete polynomial simulator (5), continuous time is also removed, and the step size becomes the price paid for its absence.

Related work. Our work sits at the intersection of three traditions. Classical *subrecursive function theory* characterizes complexity classes by restricting recursion schemes – Cobham for P [21], Bellantoni–Cook [5], Leivant–Marion for FPSPACE [28], see [20] for a survey – but always takes composition as a built-in closure. The *neural-computation* line [38, 37, 4, 29] showed early on that recurrent analog systems compute through state update, feedback, and thresholding rather than symbolic composition. The *GPAC and polynomial-ODE* tradition, from the Grzegorczyk-level analog characterizations of [18, 19, 15, 24] to the polynomial-time and polynomial-space characterizations of [13, 33, 22, 11] and the discrete-ODE lines of [9, 10, 1, 2, 3], shows that polynomial dynamics provide a machine-independent framework over the reals. Our paper combines these: machine-free like implicit complexity, polynomial dynamics like the ODE literature, discrete-time and raw-input like neither, and the internalization of composition as a consequence of the dynamics rather than a postulate.

2 The asymmetry between discrete and continuous polynomial dynamics

Computation by a polynomial dynamical system. Before stating the results of this section, we make precise the common shape of computation shared by items (4) and (5) of the main theorem, and by the dynamical reading of items (2) and (3) as well. The data is a polynomial map $p : \mathbb{R}^m \to \mathbb{R}^m$, viewed as a fixed vector field on the state space $\mathbb{R}^m$. This single object can be evolved in two ways. In discrete time, it defines the iteration $x_{n+1} = p(x_n)$. In continuous time, it defines the autonomous ODE $\dot{x} = p(x)$. In both cases a computation on input $x \in \mathbb{N}^d$ is a trajectory started from the raw initial state $(x, 0, \ldots, 0)$, and the output is read off a designated coordinate at some time n (respectively t) that depends on the input.

What is striking, and what this section is about, is that these two evolutions of the same polynomial data are not computationally interchangeable. Continuous time makes available mechanisms, such as contraction toward the integers and autonomous phase switching, that a discrete iteration of the same kind of object provably cannot realize uniformly. Throughout this section, $f^{[n]} := f \circ \cdots \circ f$ (n times) denotes the n-fold iterate of a map f, with $f^{[0]} := \mathrm{id}$.

Two primitives of dynamical computation. To make the asymmetry quantitative, we examine two operations that are natural building blocks for any computation on integer inputs carried out by a dynamical system. The first is *rounding*: contracting a real state that is close to an integer back onto that integer, so that integer values can be maintained along a long trajectory. The second is *phase selection*: switching autonomously between phases of the computation after a number of steps supplied as input. Both are ubiquitous in the GPAC and polynomial-ODE literature, where they serve as the foundation on which clocks, error correction, and bounded recursion are built [23].

We now state two impossibility results showing that both of these operations, which are essentially free in continuous time for polynomial vector fields, are provably unattainable by pure polynomial maps in discrete time. The proofs are short and direct, but the consequence is conceptual: the discrete and continuous settings rest on genuinely different assumptions about what is elementary, and this explains why additional computational resources (the ρ-activation gate in the neural models, the step-size parameter in the discrete simulator) become mandatory when passing from one world to the other.

▶ Remark 3. We focus here on the contrast between discrete-time and continuous-time polynomials, but the conclusions bear directly on the neural-network setting. The choice of activation function is itself a commitment about which operations are taken as primitive. Our impossibility results make this commitment explicit: the activation ρ is not a convenience but a necessary compensation for what polynomials alone cannot do in discrete time. Since affine maps are particular polynomial maps, any impossibility for polynomial maps *a fortiori* rules out affine maps, and thus pure feedforward linear networks, as well.

Uniform rounding. One basic operation when passing between integers and reals is rounding. As observed in [23], in continuous time, this has an elegant and essentially costless solution: The map

$$\sigma(x) = x - \frac{1}{2\pi}\sin(2\pi x)$$

fixes every integer and contracts every real number toward the nearest integer: for any $\varepsilon < 1/2$, there is a contraction factor $\lambda = \lambda(\varepsilon) < 1$ such that $|\sigma(n+\delta) - n| \le \lambda\,|\delta|$ whenever $|\delta| \le \varepsilon$. Since the sine function is natively generated in the polynomial ODE world by $y'' = -y$, this contraction is available at no additional cost. This mechanism, introduced as the very first statement (Lemma 1) of [23], is the foundational building block of the entire GPAC computation theory: all subsequent control structures, such as clocks, holding phases, error correction, are built on top of it in [23].

In discrete time, the analogous task is provably impossible from a fixed polynomial map:

▶ **Proposition 4** (No uniform polynomial rounder in discrete time). *Let $0 < \varepsilon < \delta < 1/2$. There is no polynomial $p : \mathbb{R} \to \mathbb{R}$ and no integer $N \ge 1$ such that for every $m \in \mathbb{Z}$ and every u with $|u - m| \le \delta$, $|p^{[N]}(u) - m| \le \varepsilon$.*

▶ Remark 5. Here "rounding" means contracting, for every integer m simultaneously, the whole band $\{u : |u - m| \le \delta\}$ into the tighter band $\{u : |u - m| \le \varepsilon\}$. The identity map fixes each integer but contracts nothing, so it already fails as soon as $\varepsilon < \delta$; the statement asserts that no fixed polynomial, even iterated, does better. Since $p^{[N]}$ is again a polynomial, the case $N = 1$ already implies the general one, and we keep N only to stress that iterating a fixed map does not help.

Proof. Suppose such p and N exist. The polynomial $q(x) = p^{[N]}(x) - x$ satisfies $|q(m)| \le \varepsilon$ for every integer m. A polynomial bounded on infinitely many integers is constant: $q \equiv c$. But then $|\delta + c| \le \varepsilon$ and $|-\delta + c| \le \varepsilon$, which is impossible when $\varepsilon < \delta$. ◀

To achieve approximate rounding in discrete time, one can approximate σ on a bounded interval $[-B, B]$ by a polynomial whose degree depends on B and on the target accuracy. An operation that is free in continuous time becomes structurally expensive in discrete time: the polynomial degree is the currency in which the discrete construction pays.

Exact phase selection. A second fundamental operation is switching between phases of a computation. Given a counter C supplied as input, one needs an indicator that outputs 1 for the first C steps and 0 thereafter. In continuous time, a polynomial ODE comparator achieves this autonomously, driving a register exponentially fast toward 0 or 1 depending on the sign of its input. This can be done using a polynomial map [23].

▶ **Proposition 6** (No exact polynomial phase selector)**.** *There is no polynomial map $P : \mathbb{R}^m \to \mathbb{R}^m$ and coordinate projection $\pi : \mathbb{R}^m \to \mathbb{R}$, $(z_1, \dots, z_m) \mapsto z_i$, such that for every integer $C \in \mathbb{N}$,*

$$\pi(P^{[t]}(C, 0, \dots, 0)) = \begin{cases} 1 & \text{if } t < C, \\ 0 & \text{if } t \ge C, \end{cases} \qquad \text{for all } t \in \mathbb{N}.$$

Proof. For fixed t, the function $Q_t(C) = \pi(P^{[t]}(C, 0, \dots, 0))$ is a polynomial in C. Since $Q_t(C) = 1$ for all integers $C > t$, the polynomial $Q_t - 1$ has infinitely many roots, so $Q_t \equiv 1$. But $Q_t(t) = 0$ by hypothesis – a contradiction. ◀

3 Formal framework and equivalence for hard models

We now define the five dynamical models, state the equivalence theorems, and sketch the proof architecture.

Notation. Throughout the paper, an *affine map* from $\mathbb{R}^p$ to $\mathbb{R}^q$ is a function $A : \mathbb{R}^p \to \mathbb{R}^q$ of the form $A(u) = Mu + b$ with $M \in \mathbb{Q}^{q \times p}$ and $b \in \mathbb{Q}^q$. Unless stated otherwise, all affine maps considered in this paper have rational coefficients. Given an ordered tuple of distinct indices $I = (i_1, \dots, i_e) \in \{1, \dots, m\}^e$, we write $\pi_I : \mathbb{R}^m \to \mathbb{R}^e$ for the associated coordinate projection

$$\pi_I(u_1, \dots, u_m) = (u_{i_1}, \dots, u_{i_e}).$$

Where the index tuple is clear from context, we abbreviate π_I by π_e. For $u = (u_1, \dots, u_m) \in \mathbb{R}^m$ we write $\|u\|_\infty = \max_i |u_i|$ for the supremum norm, and $B_\infty(c, r) = \{z : \|z - c\|_\infty \le r\}$ for the closed ball of radius r centred at c in this norm.

3.1 The discrete exact model: recurrent ReLU computation

▶ **Definition 7** (Feedforward ReLU block). *A* feedforward ReLU block *on* $\mathbb{R}^m$ *is a map* $R : \mathbb{R}^m \to \mathbb{R}^m$ *of the form* $R = A_L \circ \sigma \circ A_{L-1} \circ \sigma \circ \cdots \circ A_1 \circ \sigma \circ A_0$, *where* $L \geq 0$ *is a fixed integer (the* depth*); the* A_ℓ *are affine maps with rational coefficients* $A_\ell : \mathbb{R}^{m_\ell} \to \mathbb{R}^{m_{\ell+1}}$, *with* $m_0 = m_{L+1} = m$ *and arbitrary intermediate widths* $m_1, \ldots, m_L \in \mathbb{N}$; *and* σ *denotes the coordinatewise application of* $\mathrm{ReLU}(t) := \max(t, 0)$.

Equivalently, R is the function computed by a feedforward neural network with L hidden layers, rational weights and biases, and ReLU activations. The map R is piecewise affine with rational pieces.

▶ **Definition 8** (Recurrent ReLU computation). *A function* $f : \mathbb{N}^d \to \mathbb{N}^e$ *admits a* recurrent ReLU computation *if there exist an integer* $m \geq d$, *a feedforward ReLU block* $R : \mathbb{R}^m \to \mathbb{R}^m$, *a primitive recursive observation time* $T : \mathbb{N}^d \to \mathbb{N}$, *and an ordered tuple of designated output-coordinate indices* $I \in \{1, \ldots, m\}^e$ *such that, for every input* $x \in \mathbb{N}^d$, *the orbit* $z_0(x) := (x, 0, \ldots, 0) \in \mathbb{R}^m$, $z_{n+1}(x) := R(z_n(x))$ *satisfies* $f(x) = \pi_I\big(z_{T(x)}(x)\big)$.

In other words: a single fixed piecewise-affine map with rational coefficients is iterated from the raw input $(x, 0, \ldots, 0)$, and the exact value of $f(x)$ is read off by projecting onto the output coordinates after $T(x)$ steps.

3.2 The neural network model: recurrent ρ-computation

For bounded-range activations (sigmoid, tanh, ...), the compact state space cannot accommodate arbitrarily large integers as raw values. We fix the encoding $\nu : \mathbb{N} \to (0, 1]$ defined by $\nu(n) := 2^{-n}$, and write $\nu(\mathbf{x}) := (2^{-x_1}, \ldots, 2^{-x_d})$ for a tuple $\mathbf{x} \in \mathbb{N}^d$. Note that $\nu(0) = 1$ and $\nu(n) \leq \frac{1}{2}$ for $n \geq 1$.

▶ **Definition 9** (Feedforward ρ-block). *Let* $\rho : [0, 1] \to [0, 1]$. *A* feedforward ρ-block *of width* m *is a map* $R : [0, 1]^m \to [0, 1]^m$ *of the form* $R = A_L \circ \rho \circ A_{L-1} \circ \rho \circ \cdots \circ A_1 \circ \rho \circ A_0$, *where* $L \geq 0$ *is a fixed integer depth; the* A_ℓ *are affine maps with rational coefficients* $A_\ell : \mathbb{R}^{m_\ell} \to \mathbb{R}^{m_{\ell+1}}$, *with* $m_0 = m_{L+1} = m$ *and arbitrary intermediate widths, satisfying* $A_\ell\big([0, 1]^{m_\ell}\big) \subseteq [0, 1]^{m_{\ell+1}}$ *for every* ℓ; *and* ρ *denotes the coordinatewise application of* ρ *to a vector of* $[0, 1]^{m_\ell}$.

The inclusion condition on the A_ℓ simply ensures that the composition is well-defined on $[0, 1]^m$ and takes values in $[0, 1]^m$.

▶ **Definition 10** (Admissible activation). *A function* $\rho : [0, 1] \to [0, 1]$ *is* admissible *if:*

- **(i)** *ρ is computable in the sense of computable analysis [40, 27], with a primitive recursive evaluation oracle and a primitive recursive modulus of uniform continuity;*
- **(ii)** *there exist a feedforward ρ-block $Z : [0, 1] \to [0, 1]$ and some $\eta \in (0, \frac{1}{8})$ such that $|Z(u)| \leq \eta$ for $u \in [0, \frac{5}{8}]$ and $|Z(u) - 1| \leq \eta$ for $u \in [\frac{7}{8}, 1]$.*

Condition (i) ensures primitive recursive simulability. Condition (ii) provides a robust zero-detection mechanism: the network can distinguish $\nu(0) = 1$ from all other codes. Both conditions hold for all standard sigmoid activations.

▶ **Definition 11** (Recurrent ρ-computation). *Let* ρ *be admissible. A function* $f : \mathbb{N}^d \to \mathbb{N}^e$ *admits a* recurrent ρ-computation *if there exist an integer* $m \geq d$, *a feedforward* ρ*-block* $R : [0, 1]^m \to [0, 1]^m$, *a primitive recursive observation time* $T : \mathbb{N}^d \to \mathbb{N}$, *a primitive recursive output precision* $S : \mathbb{N}^d \to \mathbb{N}$ *with* $S(x) \geq 2$, *and an ordered tuple* $I = (i_1, \ldots, i_e) \in \{1, \ldots, m\}^e$ *of designated output-coordinate indices, such that the orbit* $z_0(x) := (\nu(x), 0, \ldots, 0) \in [0, 1]^m$, $z_{n+1}(x) := R(z_n(x))$ *satisfies, for every output index* $j \in \{1, \ldots, e\}$ *and every* $x \in \mathbb{N}^d$: $\Big|\big(\pi_I(z_{T(x)}(x))\big)_j - \nu\big(f_j(x)\big)\Big| \leq 2^{-S(x)}\,\nu\big(f_j(x)\big)$.

The bound $S \geq 2$ guarantees unique decodability: for every $n \in \mathbb{N}$, the nearest ν-code to $\nu(n)$ is at distance $\frac{1}{2}\nu(n)$, so a relative error of $2^{-S} \leq \frac{1}{4}$ suffices. The precision function $S(x)$ plays the role of a readout resource: the network and initial state are fixed once x is given, and only the required output precision varies with the input.

3.3 The continuous model: polynomial ODE computation

Here and below, a *polynomial map* (resp. *polynomial vector field*) from $\mathbb{R}^p$ to $\mathbb{R}^q$ is a function whose q components are polynomials in the p input variables with rational coefficients. Thus every affine map is in particular a polynomial map.

▶ **Definition 12** (Robust polynomial ODE representation). *A function $f : \mathbb{N}^d \to \mathbb{N}^e$ admits a* robust polynomial ODE representation *if there exist an integer $M \geq d$, a polynomial vector field $F : \mathbb{R}^M \to \mathbb{R}^M$ with rational coefficients, a primitive recursive observation time $\tau : \mathbb{N}^d \to \mathbb{N}$, a primitive recursive safety bound $\widetilde{B} : \mathbb{N}^d \to \mathbb{N}$, and an ordered tuple $I = (i_1, \ldots, i_e) \in \{1, \ldots, M\}^e$ of designated output-coordinate indices, such that for every $x \in \mathbb{N}^d$ the solution $Y = Y_x$ of the Cauchy problem*

$$Y'(t) = F\big(Y(t)\big), \qquad Y(0) = (x, 0, \ldots, 0) \in \mathbb{R}^M,$$

satisfies (i) Y is defined on $[0, \tau(x)+1]$; (ii) $\|Y(t)\|_\infty \leq \widetilde{B}(x)$ on this interval; (iii) $\big\|\pi_I(Y(t)) - f(x)\big\|_\infty < \frac{1}{2}$ for all $t \in [\tau(x), \tau(x)+1]$.

The vector field F is fixed; only the observation time, the safety bound, and the choice of output indices depend on the represented function f. The output is recovered by coordinate-wise rounding of $\pi_I(Y(\tau(x)))$ to the nearest integer: condition (iii) guarantees that every output coordinate stays within distance $\frac{1}{2}$ of the integer $f_j(x)$ throughout the entire window $[\tau(x), \tau(x)+1]$, so the reading is insensitive to the exact observation time. This is the sense in which the representation is *robust*.

3.4 The intermediate model: step-size-controlled polynomial simulation

▶ **Definition 13** (Step-size-controlled polynomial representation). *A function $f : \mathbb{N}^d \to \mathbb{N}^e$ admits a* step-size-controlled polynomial representation *if there exist an integer $M \geq d+1$, a polynomial map $P : \mathbb{R}^M \to \mathbb{R}^M$ with rational coefficients, a primitive recursive precision threshold $S : \mathbb{N}^d \to \mathbb{N}$, a primitive recursive observation count $N : \mathbb{N}^d \times \mathbb{N} \to \mathbb{N}$, and an ordered tuple $I = (i_1, \ldots, i_e) \in \{1, \ldots, M\}^e$ of designated output-coordinate indices, such that for every $x \in \mathbb{N}^d$ and every precision parameter $s \in \mathbb{N}$ with $s \geq S(x)$, the orbit defined by $Z_0 := (x,\, 2^{-s},\, 0, \ldots, 0) \in \mathbb{R}^M$, $\quad Z_{n+1} := P(Z_n)$ satisfies $\big\|\pi_I(Z_{N(x,s)}) - f(x)\big\|_\infty < \frac{1}{2}$.*

We call a model *hard* when every orbit started at an integer state stays in $\mathbb{Z}^M$ (the ReLU model and the threshold-affine normal form below), and *smooth* otherwise, that is, when the orbit generically takes non-integer values (the ρ-model, the polynomial ODE, and the step-size-controlled simulator).

The map P is purely polynomial: no threshold gate or transcendental activation is involved. The step size $h = 2^{-s}$ is supplied as part of the initial state, stored in a dedicated coordinate.

We isolate the proof-level normal form through which all equivalences pass. Fix $\Theta : \mathbb{R} \to \mathbb{R}$ with $\Theta(u) = 0$ for $u \leq 0$ and $\Theta(u) = 1$ for $u \geq 1$; its values on $(0, 1)$ are irrelevant here, since all gate arguments will be integers.

▶ **Definition 14** (Threshold-affine normal form). *A function $f : \mathbb{N}^d \to \mathbb{N}^e$ admits a* threshold-affine normal form *if there exist an integer m, affine maps $A_0, \dots, A_s : \mathbb{R}^m \to \mathbb{R}^m$ and $q_1, \dots, q_s : \mathbb{R}^m \to \mathbb{R}$* with integer coefficients*, a primitive recursive time bound $T : \mathbb{N}^d \to \mathbb{N}$, and output coordinates $\pi_e : \mathbb{R}^m \to \mathbb{R}^e$ such that, writing*

$$P(y) \;=\; A_0(y) \;+\; \sum_{j=1}^{s} \Theta(q_j(y))\, A_j(y), \tag{1}$$

and defining $y_0(x) := (x, 0, \dots, 0)$, $y_{n+1}(x) := P(y_n(x))$, one has $f(x) = \pi_e(y_{T(x)}(x))$ for every $x \in \mathbb{N}^d$.

Since the A_j and q_j have integer coefficients and $y_0(x) \in \mathbb{Z}^m$, every iterate $y_n(x)$ lies in $\mathbb{Z}^m$ and every gate argument $q_j(y_n(x))$ is an integer; in particular, Θ is never evaluated in its ambiguous region $(0, 1)$.

▶ **Theorem 15** (Main equivalence: hard models). *For a function $f : \mathbb{N}^d \to \mathbb{N}^e$, the following are equivalent:*

1. *f is primitive recursive;*
2. *f admits a recurrent ReLU computation;*
3. *f admits a threshold-affine normal form.*

The proof is based on the fact that primitive recursive functions are known to coincide with those computable by a LOOP program [30], and that a single LOOP instruction compiles directly into the form given by Equation (1). The interest of the theorem lies not in the proof but in what it singles out: the threshold-affine normal form of Item 3. This is the central object of the paper, and the hinge through which the smooth models will also be analyzed.

More details on proof architecture. (1) ⇒ (3) (Section 5). A LOOP program computing f is compiled one instruction at a time into (1): arithmetic is carried by the affine maps A_j, and zero-testing by the threshold Θ applied to an integer argument $q_j(y)$. (3) ⇒ (2). Each threshold gate $\Theta(q_j(y))$ is replaced by $\mathrm{ReLU}(q_j(y)) - \mathrm{ReLU}(q_j(y) - 1)$, an expression that coincides with Θ on every integer argument. (2) ⇒ (1). A fixed ReLU block with rational parameters induces, on rational initial states, a trajectory whose coordinates are primitive recursive in the iteration count.

A subtlety. The compilation (3) ⇒ (2) produces a ReLU block that agrees with the underlying threshold-affine map only along the integer-valued trajectories relevant to the computation, not on all real inputs. This is enough here, since the equivalence concerns functions on $\mathbb{N}^d$.

4 Equivalence for smooth models

4.1 A dynamical systems perspective

The equivalences for hard models in Theorem 15 establish more than the bare statement. The constructions endow the threshold-affine map P of (1), together with its orbits, with four structural properties that are most naturally phrased in the language of dynamical systems.

(i) *The integer lattice is forward-invariant under P.* The affine maps A_j have integer coefficients, and Θ takes values in $\{0,1\}$ on integer arguments; hence $P(\mathbb{Z}^m) \subseteq \mathbb{Z}^m$. Every trajectory issued from the raw integer initial condition $y_0(x) = (x, 0, \ldots, 0)$ therefore stays on the lattice.

(ii) *The reference trajectory is bounded, uniformly in the input.* There exists a primitive recursive function $B : \mathbb{N}^d \to \mathbb{N}$ such that, for every $x \in \mathbb{N}^d$ and every $n \leq T(x)$, the state $y_n(x)$ lies in the cube $[-B(x), B(x)]^m$. Combined with (i), this confines the effective dynamics to a finite subset of the lattice $\mathbb{Z}^m$, whose size is primitive recursively controlled by the input.

(iii) *A distinguished orbit carries the computational content.* For each input x, the finite sequence $y_0(x), \ldots, y_{T(x)}(x)$ is a specific trajectory of P along which the output is read off at time $T(x)$. The computation is encoded by the family of these *reference orbits*, indexed by $x \in \mathbb{N}^d$.

(iv) *The reference orbit is robust to perturbations of the threshold.* Every gate argument $q_j(y_n(x))$ on the reference orbit is an integer; it lies in the region where Θ is fully determined ($u \leq 0$ or $u \geq 1$) and never enters the ambiguous interval $(0, 1)$. The orbit is therefore insensitive to any modification of Θ that preserves its values on $\mathbb{Z}$.

Property (iv) is the hinge between the hard and smooth worlds: any alternative dynamics agreeing with P on $\mathbb{Z}^m$ reproduces the computation faithfully. Formally, we have:

▶ **Theorem 16** (Uniform-robust characterization). *A function $f : \mathbb{N}^d \to \mathbb{N}^e$ is primitive recursive if and only if it admits a threshold-affine normal form in which $\Theta(u) = 0$ for $u \leq \frac{1}{4}$ and $\Theta(u) = 1$ for $u \geq \frac{3}{4}$.*

Principle of the proof: the margin condition is strictly stronger than the threshold condition of Definition 14, so the "if" direction is immediate from Theorem 15. For the "only if" direction, the compilation of that theorem produces integer gate arguments (property (iv)), so any threshold agreeing with Θ on $\mathbb{Z}$, in particular the margin version $\widetilde{\Theta}(u) = \mathrm{ReLU}(2u - \frac{1}{2}) - \mathrm{ReLU}(2u - \frac{3}{2})$, leaves the reference orbit unchanged.

Properties (i)–(iv) motivate the smooth models of the next subsection: it is enough for the smooth dynamics to send each integer state into the *basin of correct decoding* of its successor, on the bounded trapping region.

4.2 Integer-faithful shadowing

We formalize required concepts using a notion of shadowing.

▶ **Definition 17** (Integer-faithful approximation). *Let $D \subseteq \mathbb{R}^m$, let $F, \widetilde{F} : D \to \mathbb{R}^m$, and assume $F(D \cap \mathbb{Z}^m) \subseteq \mathbb{Z}^m$. The map $\widetilde{F}$ is* integer-faithful to F on D *if $\|\widetilde{F}(y) - F(y)\|_\infty < \frac{1}{2}$ for every $y \in D \cap \mathbb{Z}^m$.*

The map $\widetilde{F}$ need not preserve the lattice or coincide with F on non-integer points. It only needs to land in the correct decoding basin on the integer skeleton of D.

▶ **Lemma 18** (Decoded shadowing). *Under the hypotheses above, if additionally $F(D \cap \mathbb{Z}^m) \subseteq D \cap \mathbb{Z}^m$, then the decoded orbit $\widetilde{y}_{n+1} = \mathrm{round}(\widetilde{F}(\widetilde{y}_n))$ started from $\widetilde{y}_0 = y_0 \in D \cap \mathbb{Z}^m$ satisfies $\widetilde{y}_n = y_n$ for all n such that $y_0, \ldots, y_n \in D$.*

Proof. Induction: if $\widetilde{y}_n = y_n \in D \cap \mathbb{Z}^m$, then $\widetilde{F}(y_n)$ lies within $\frac{1}{2}$ of $F(y_n) \in \mathbb{Z}^m$, so rounding recovers y_{n+1}. ◀

For the smooth models, that do not round exactly between steps, the integer-faithful property alone (Definition 17) is no longer sufficient. What we need in addition is a quantitative control on how a one-step error propagates along the orbit. The following discrete analogue of Grönwall's inequality provides it.

▶ **Lemma 19** (Discrete Grönwall shadowing). *Let $D \subseteq \mathbb{R}^m$ and let $P : D \to D$ admit an orbit $y_0, y_1, \ldots, y_T$ (so $y_{n+1} = P(y_n)$ for $0 \le n \le T-1$). Let $L \ge 0$, $\varepsilon \ge 0$, $r > 0$ satisfy the tube condition $\varepsilon \sum_{k=0}^{T-1} L^k \le r$, and let Φ be defined at least on $\bigcup_{n=0}^{T-1} B_\infty(y_n, r)$ and satisfy, for every $n \in \{0, \ldots, T-1\}$ and every $z \in B_\infty(y_n, r)$, $\|\Phi(z) - P(y_n)\|_\infty \le L\,\|z - y_n\|_\infty + \varepsilon$.*

Then the orbit defined by $z_0 := y_0$ and $z_{n+1} := \Phi(z_n)$ is well-defined for $0 \le n \le T$ and satisfies $\|z_n - y_n\|_\infty \le \varepsilon \sum_{k=0}^{n-1} L^k$ $(0 \le n \le T)$, where the empty sum at $n = 0$ is taken to be zero.

Proof sketch. Set $e_n := \|z_n - y_n\|_\infty$. Then $e_0 = 0$, and the one-step bound applied to $z = z_n$ gives the linear-plus-constant recurrence $e_{n+1} \le L\,e_n + \varepsilon$. Induction yields $e_n \le \varepsilon \sum_{k=0}^{n-1} L^k$; the tube condition guarantees $e_n \le r$ throughout, so z_n never leaves the domain where the inequality holds. ◀

4.3 Main theorem: all models

▶ **Theorem 20** (Main equivalence: hard and smooth models). *For a function $f : \mathbb{N}^d \to \mathbb{N}^e$, the following are equivalent:*

1. *f is primitive recursive;*
2. *f admits a recurrent ReLU computation;*
3. *f admits a recurrent ρ-computation;*
4. *f admits a uniform threshold-affine normal form;*
5. *f admits a polynomial ODE representation;*
6. *f admits a step-size-controlled polynomial representation.*

Proof architecture. By Theorems 15 and 16, it suffices to pass from the uniform-robust hard normal form to the smooth models and to close the loop back to primitive recursion. The guiding principle is that the hard dynamics comes with a reference orbit and a uniform margin, so each smooth model only has to stay in the correct decoding basin along that orbit.

(4) ⇒ (5) (Section 6). Each threshold becomes a polynomial ODE comparator, and each affine update is realized by continuous-time transport in a sample-and-hold cycle; correctness follows from Lemma 19 on the trapping region.

(4) ⇒ (3) (Section 7). The hard computation is transported to the ν-coded domain, and each threshold is replaced by a smooth gate built from ρ; the margin condition ensures that the smooth gate takes the same branch decisions along the reference orbit.

The remaining three implications follow standard arguments: (5) ⇒ (6) is Euler discretization on the primitive-recursively bounded time-space region; (6) ⇒ (1) uses that bounded iteration of a fixed polynomial map with primitive recursive parameters is itself primitive recursive; and (3) ⇒ (1) combines effective ρ-approximation with primitive recursive simulation of the recurrent dynamics.

5 From primitive recursion to threshold-affine normal form

This section proves (1) ⇒ (3) of Theorem 15: every primitive recursive function admits a threshold-affine normal form.

The proof uses the standard characterization of primitive recursive functions by LOOP programs [30]. A LOOP program uses registers $R_1, \dots, R_s$ with atomic instructions $R_i := 0$, $R_i := R_j$, $R_i := R_j + 1$, composed sequentially and by bounded loops `for` R_i `do` Q. A function is primitive recursive if and only if it is computed by a fixed LOOP program, and the running time of a fixed LOOP program is itself primitive recursive [30, 20].

The only control construct is the bounded loop `for` R_i `do` Q, whose iteration count is read once from R_i on entry and cannot be modified inside Q. This is the single feature separating LOOP programs from unlimited register machines, which additionally allow conditional jumps, hence unbounded `while` loops, and therefore compute every partial recursive function [35]; lacking unbounded iteration, LOOP programs capture exactly the (total) primitive recursive functions.

From Θ , and its properties, we can derive an exact zero-test: $\mathrm{Zero}(u) := 1 - \Theta(u) - \Theta(-u)$, which satisfies $\mathrm{Zero}(u) = 1$ iff $u = 0$ for $u \in \mathbb{Z}$.

Compiling one step into a threshold-affine map. Fix a LOOP program Π computing $f : \mathbb{N}^d \to \mathbb{N}^e$. Since Π is fixed, the number of registers, instructions, and the maximal nesting depth are all constants. We encode a complete configuration by a vector $y \in \mathbb{N}^m$ containing the data registers, a one-hot program counter, a finite loop-counter stack, and a few auxiliary registers. The initial configuration for input x is $(x, 0, \dots, 0)$.

Each instruction step updates the configuration as follows. Assignments ($R_i := 0$, $R_i := R_j$, $R_i := R_j + 1$) are affine. Loop entry and exit require testing whether a counter is zero or positive, which is expressed exactly by Zero and Θ on integer-valued registers. The program counter, stored in one-hot Boolean form, is combined using affine operations and Θ (for instance, $a \wedge b = \Theta(a + b - 1)$). Since all quantities are fixed and finite, the global one-step transition is $P(y) = A_0(y) + \sum_{j=1}^{s} \Theta(q_j(y))\, A_j(y)$: a threshold-affine normal form.

Notice that the nonlinearity of the construction comes entirely from the threshold tests on integer control registers; the arithmetic updates themselves are affine.

6 From uniform threshold-affine normal form to polynomial ODE

We now prove (4) ⇒ (5) of Theorem 20.

The strategy. Let $P(y) = A_0(y) + \sum_{j=1}^{s} \Theta(q_j(y))\, A_j(y)$ be a uniform threshold-affine normal form with reference orbit $y_0(x), \dots, y_{T(x)}(x) \subseteq \mathbb{Z}^m \cap [-B(x), B(x)]^m$, integer gate arguments (property (iv) of Section 4.1), and output projection π_I.

Our goal is a polynomial vector field F_P whose flow samples, at a fixed period $\kappa_0 > 0$, a sequence of states z_n that tracks $y_n(x)$ to within a constant fraction of the unit spacing of $\mathbb{Z}^m$. This is exactly the sort of tracking that Section 4.2 calls *integer-faithful shadowing*, applied at a resolution that allows decoding by rounding to the nearest integer. Concretely, we will build F_P so that the sampled map $z_n \mapsto z_{n+1}$ agrees with P up to a contraction-plus-noise error of the form $\|z_{n+1} - P(z_n)\|_\infty \le \lambda \|z_n - y_n(x)\|_\infty + \eta$, with $\lambda < 1$ and $\eta/(1-\lambda) < \frac{1}{4}$. Since P is integer-faithful to itself, the discrete Grönwall shadowing lemma (Lemma 19) then keeps the sampled orbit within the $\frac{1}{4}$-decoding basin of the reference orbit throughout time.

The substance of the section is that two standard polynomial-ODE gadgets suffice to produce such a sampled map: a *smooth clock*, which organizes the cycle, and a *polynomial comparator*, which realizes Θ up to the η-noise tolerated by the shadowing lemma.

▶ Remark 21. The resulting construction can be read as a particular improved instance of the Branicky [17] alternating-targeting trick used, for example, in [23, 11]. We provide an original view on it. Rather than analyzing the continuous flow step by step, we treat it

as a candidate integer-faithful approximation of P in the sense of Section 4.2 and let the discrete Grönwall lemma handle the propagation of error. The one computation that remains specific to the continuous construction is the per-cycle estimate (2); everything else is a direct application of the shadowing framework.

The smooth clock. Let (c_1, c_2) satisfy the rational-coefficient ODE $c_1' = -c_2$, $c_2' = c_1$ with initial condition $(c_1(0), c_2(0)) = (1, 0)$; its solution $(c_1(t), c_2(t)) = (\cos t, \sin t)$ is periodic of period $\kappa_0 := 2\pi$. A polynomial driver $h(t)$, computed as a fixed polynomial expression in (c_1, c_2), can be arranged to be non-negative, to satisfy $h(t) \geq 1$ on each *compute* half-period $[n\kappa_0,\, n\kappa_0 + \kappa_0/2]$, and to vanish on each *hold* half-period $[n\kappa_0 + \kappa_0/2,\, (n+1)\kappa_0]$, for every integer $n \geq 0$ (see [11, §3] for a standard realization). Everything below is phrased as an ODE whose right-hand side multiplies $h(t)$: active during compute phases, frozen during hold phases.

The polynomial comparator. For every $\delta \in (0, \frac{1}{4})$ and every contraction rate $\Lambda > 0$, there is a polynomial vector field on an extra coordinate r such that for every input $u \in \mathbb{R}$ with $u \notin (\frac{1}{4}, \frac{3}{4})$ and every compute half-period $[t_0, t_0 + \kappa_0/2]$, the driven equation $r' = h(t) \cdot \Lambda \cdot (\Theta_\infty(u) - r)$ satisfies $|r(t_0 + \kappa_0/2) - \Theta_\infty(u)| \leq e^{-\Lambda\kappa_0/2}\, |r(t_0) - \Theta_\infty(u)|$, which is below δ once Λ is chosen large enough. Here $\Theta_\infty(u) \in \{0, 1\}$ is the target value determined by whether $u \leq \frac{1}{4}$ or $u \geq \frac{3}{4}$; it is computed by a polynomial in u approximating Θ on the two separated regions (details in [11, §3]). In our application u will be one of the gate arguments $q_j(\pi(Z))$, tracked continuously by an affine expression in the state. Property (iv) of Section 4.1 guarantees that every comparator call on the reference orbit satisfies the margin hypothesis $q_j(y_n(x)) \notin (\frac{1}{4}, \frac{3}{4})$, hence falls in a determined regime.

Composing the step. Given a cycle-start state $z_n \in \mathbb{R}^m$ close to $y_n(x)$, we want a cycle-end state z_{n+1} close to $P(y_n(x))$. Per clock period κ_0 we allocate s comparator coordinates $r_1, \ldots, r_s$ driven by the comparator ODE on $u_j := q_j(\pi(Z))$ (so $r_j \to \Theta(q_j(y_n(x)))$ exponentially during the compute phase); m target-tracking coordinates $w_1, \ldots, w_m$ driven by the sample-and-hold equation $w_k' = h(t) \cdot \Lambda \cdot \big((A_0(\pi(Z)) + \sum_j r_j\, A_j(\pi(Z)))_k - w_k\big)$; and a projection π that reads $w(t)$ at the end of each hold phase. Assembling these ODEs with the clock gives a polynomial vector field F_P with rational coefficients on $\mathbb{R}^M$, $M := 2 + s + m$, with initial condition $(1, 0, 0, \ldots, 0, x, 0, \ldots, 0)$. An explicit one-period calculation yields, with $\delta := e^{-\Lambda\kappa_0/2}$, $\lambda := \delta \cdot \max_j \|A_j\|_\infty$, and $\eta := \delta\,(1 + s \cdot \max_j \|A_j\|_\infty)$, the estimate

$$\|z_{n+1} - P(y_n(x))\|_\infty \leq \lambda\, \|z_n - y_n(x)\|_\infty + \eta, \qquad z_n := w(n\kappa_0). \tag{2}$$

Choosing Λ large enough makes $\lambda < 1$ and $\eta/(1-\lambda) < \frac{1}{4}$.

▶ **Theorem 22.** *Every uniform threshold-affine normal form admits a robust polynomial ODE representation.*

Proof sketch. Set $e_n := \|\pi(Z(n\kappa_0)) - y_n(x)\|_\infty$. By (2), $e_0 = 0$ and $e_{n+1} \leq \lambda e_n + \eta$; Lemma 19 with $L := \lambda$, $\varepsilon := \eta$, $r := \frac{1}{4}$ gives $e_n < \frac{1}{4}$ for every $n \leq T(x)$, so the sampled state is integer-faithful to the reference orbit. Setting $\tau(x) := \lceil \kappa_0 T(x) \rceil$ and $\widetilde{B}(x) := R_P(B(x)+1)$ (a polynomial Grönwall majorant for F_P on one cycle), the interval $[\tau(x), \tau(x)+1]$ lies inside the final hold half-period, on which w is frozen within $\frac{1}{4}$ of $y_{T(x)}(x)$. Projecting by π_I yields $\|\pi_I(Z(t)) - f(x)\|_\infty < \frac{1}{4}$ throughout $[\tau(x), \tau(x)+1]$, which is Definition 12. ◀

7 From uniform threshold-affine normal form to recurrent ρ-computation

We now prove (4) $\Rightarrow$ (3) of Theorem 20. Let $P(y) = A_0(y) + \sum_{j=1}^{s} \Theta(q_j(y))\, A_j(y)$ be a uniform threshold-affine normal form on $\mathbb{R}^m$, with A_j and q_j of integer coefficients (as produced by Theorem 15), reference orbit $y_0(x), \dots, y_{T(x)}(x) \subseteq \mathbb{Z}^m \cap [-B(x), B(x)]^m$ (properties (i)–(ii) of Section 4.1), and output coordinates π_I. We use a strategy with similarities with previous section, except that basins are not anymore corresponding to integers, but to encoded integers.

Encoding and basins. Encode integers by $\nu(n) = 2^{-|n|}$, with the sign carried on an auxiliary coordinate (suppressed in the notation below; the construction is symmetric). For $y \in \mathbb{Z}^m$ write $\lfloor y \rfloor_\nu := (\nu(y_1), \dots, \nu(y_m)) \in [0,1]^m$, and define the *$\varepsilon$-basin* around $\lfloor y \rfloor_\nu$ as the product of relative intervals $\mathcal{B}_\varepsilon(y) := \prod_{k=1}^{m} [(1-\varepsilon)\nu(y_k), (1+\varepsilon)\nu(y_k)] \subseteq [0,1]^m$. The relative tolerance is the natural one for dyadic codes, and $\varepsilon < \frac{1}{8}$ ensures distinct states have disjoint basins.

The coded one-step update. The three ingredients of P transport to the coded domain as follows. Integer-coefficient affine maps transport to feedforward ρ-blocks with dyadic-rational coefficients (an integer multiplier c acts on $\nu(n)$ by $\nu(n) \mapsto 2^{-c}\nu(n)$). The threshold $\Theta(q_j(y)) \in \{0,1\}$ is replaced by $Z(\check{q}_j(z))$, where Z is the admissible detector of Definition 10 and $\check{q}_j$ is the coded form of q_j; property (iv) of Section 4.1 guarantees that $\nu(q_j(y_n)) \in [0, \frac{5}{8}] \cup [\frac{7}{8}, 1]$ throughout the reference orbit, the region where Z is unambiguous. Writing $\check{A}_j$ for the coded form of A_j, the global coded one-step update is

$$R(z) \;=\; \check{A}_0(z) + \sum_{j=1}^{s} Z\big(\check{q}_j(z)\big) \cdot \check{A}_j(z), \tag{3}$$

which is a feedforward ρ-block as required by Definition 9.

▶ **Lemma 23** (One robust smooth coded step)**.** *There exists $\varepsilon_0 \in (0, \frac{1}{8})$, depending only on the normal form and on the detector margin η, such that for every $\varepsilon \le \varepsilon_0$ and every $y \in \mathbb{Z}^m$ with all gate arguments $q_j(y)$ integer, $R(\mathcal{B}_\varepsilon(y)) \subseteq \mathcal{B}_\varepsilon(P(y))$.*

Proof sketch. For $z \in \mathcal{B}_\varepsilon(y)$, each coded affine map $\check{A}_j(z)$ lies in a basin of radius $O(\varepsilon)$ around $\lfloor A_j(y) \rfloor_\nu$, and each coded gate argument $\check{q}_j(z)$ differs from $\nu(q_j(y))$ by a relative error $O(\varepsilon)$. By property (iv), $\nu(q_j(y))$ lies in an unambiguous region of Z, so $Z(\check{q}_j(z))$ differs from $\Theta(q_j(y))$ by at most η. Assembling these bounds, the coordinatewise error of $R(z)$ against $\lfloor P(y) \rfloor_\nu$ is $O(\varepsilon) + O(\eta)$, with constants depending only on the normal form. Since the target basin has radius proportional to ε, fitting requires ε and η to be matched: it suffices to take ε above a fixed multiple of η (to absorb the constant η-contribution) and below a fixed upper bound (to absorb the ε-contribution). Both conditions are compatible because $\eta < \frac{1}{8}$. ◀

▶ **Theorem 24.** *Every uniform threshold-affine normal form yields a recurrent ρ-computation.*

Proof sketch. The reference orbit stays in the cube $[-B(x), B(x)]^m$ (property (ii)) and has integer gate arguments throughout (property (iv)), so Lemma 23 applies at every step with a fixed basin radius ε. Choosing $\varepsilon := 2^{-s_0(x)}$ with $s_0(x)$ primitive recursive in $B(x)$ makes the induction $z_n \in \mathcal{B}_\varepsilon(y_n(x))$ go through from the raw encoded input. At $n = T(x)$ the output coordinates approximate $\nu(f_j(x))$ within relative error $2^{-s_0(x)}$, which is Definition 11 with $T_\rho := T$ and $S_\rho := s_0$. ◀

8 Conclusion

We have characterized the primitive recursive functions as the class of functions computable by bounded iteration of a feedforward ReLU network, by bounded-range recurrent ρ-networks under ν-encoding, by robust polynomial ODEs, and by step-size controlled iteration of a polynomial map. The equivalences are mediated by a threshold-affine normal form that isolates the minimal branching structure behind a primitive recursive computation. The five formalisms realize the same class through genuinely different mechanisms, and the paper also clarifies a real asymmetry between them: exact recursive unfolding is natural in discrete time, while uniform rounding and autonomous phase organization are natural in continuous time.

The primitive recursive functions sit at the top of a rich hierarchy of classes obtained by restricting the recursion scheme: the Grzegorczyk hierarchy [26, 25], FP [21, 5], FPSPACE [28, 39], and more generally the subrecursive hierarchies surveyed in [20]. Each of our five characterizations offers a lens through which these subclasses may be redescribed, by restricting the corresponding dynamical resource (iteration count, domain diameter, precision, observation time, or degree of the polynomial map). Carrying out this programme systematically is a concrete direction for future work.

Of particular interest in this respect is the step-size-controlled polynomial model. Although polynomial, it is genuinely distinct from the discrete-ODE frameworks of [9, 10, 1, 2, 3], which take the sign function and discrete derivatives as primitive operations: our model is *strictly polynomial*, with the step size, not necessarily unitary here, playing the role of an externally supplied precision parameter. This opens a parallel route to polynomial-only characterizations of the standard complexity classes.

References

1 Melissa Antonelli, Arnaud Durand, and Juha Kontinen. A new characterization of FAC^0 via discrete ordinary differential equations. In Rastislav Královic and Antonín Kucera, editors, *49th International Symposium on Mathematical Foundations of Computer Science, MFCS 2024, Bratislava, Slovakia, August 26-30, 2024*, volume 306 of *LIPIcs*, pages 10:1–10:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.MFCS.2024.10.

2 Melissa Antonelli, Arnaud Durand, and Juha Kontinen. Characterizing small circuit classes from FAC^0 to FAC^1 via discrete ordinary differential equations. In Pawel Gawrychowski, Filip Mazowiecki, and Michal Skrzypczak, editors, *50th International Symposium on Mathematical Foundations of Computer Science, MFCS 2025, Warsaw, Poland, August 25-29, 2025*, volume 345 of *LIPIcs*, pages 10:1–10:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.MFCS.2025.10.

3 Melissa Antonelli, Arnaud Durand, and Juha Kontinen. Towards new characterizations of small circuit classes via discrete ordinary differential equations. *Theor. Comput. Sci.*, 1062:115655, 2026. doi:10.1016/J.TCS.2025.115655.

4 José L. Balcázar, Ricard Gavaldà, Hava T. Siegelmann, and Eduardo D. Sontag. Some structural complexity aspects of neural computation. In *Proceedings of the Eigth Annual Structure in Complexity Theory Conference, San Diego, CA, USA, May 18-21, 1993*, pages 253–265. IEEE Computer Society, 1993. doi:10.1109/SCT.1993.336521.

5 Stephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the poly-time functions. *Computational Complexity*, 2(2):97–110, 1992. doi:10.1007/bf01201998.

6 Manon Blanc and Olivier Bournez. Simulation of Turing machines with analytic discrete ODEs: Polynomial-time and space over the reals characterised with discrete ordinary differential equations. *Journal of Logic and Analysis*, 17, 2025. Accepted for publication. To appear. URL: http://logicandanalysis.org/index.php/jla/article/view/505.

7 Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers; NP completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, July 1989. doi:10.1142/9789812792839_0013.

8 Olivier Bournez, Manuel Lameiras Campagnolo, Daniel Silva Graça, and Emmanuel Hainry. Polynomial differential equations compute all real computable functions on computable compact intervals. *J. Complex.*, 23(3):317–335, 2007. doi:10.1016/j.jco.2006.12.005.

9 Olivier Bournez and Arnaud Durand. Recursion schemes, discrete differential equations and characterization of polynomial time computations. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 23:1–23:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.23.

10 Olivier Bournez and Arnaud Durand. A characterization of functions over the integers computable in polynomial time using discrete ordinary differential equations. *Computational Complexity*, 32(2):7, 2023. doi:10.1007/s00037-023-00240-1.

11 Olivier Bournez, Riccardo Gozzi, Daniel Silva Graça, and Amaury Pouly. A continuous characterization of PSPACE using polynomial ordinary differential equations. *Journal of Complexity*, 77:101755, August 2023. doi:10.1016/J.JCO.2023.101755.

12 Olivier Bournez, Daniel Silva Graça, and Amaury Pouly. Polynomial time corresponds to solutions of polynomial ordinary differential equations of polynomial length: The general purpose analog computer and computable analysis are two efficiently equivalent models of computations. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, Rome, Italy, July 11-15, 2016*, volume 55 of *LIPIcs*, pages 109:1–109:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. ICALP'2016 Track B, Best Paper Award. doi:10.4230/LIPIcs.ICALP.2016.109.

13 Olivier Bournez, Daniel Silva Graça, and Amaury Pouly. Polynomial time corresponds to solutions of polynomial ordinary differential equations of polynomial length. *Journal of the ACM*, 64(6):38:1–38:76, 2017. doi:10.1145/3127496.

14 Olivier Bournez, Daniel Silva Graça, Amaury Pouly, and Ning Zhong. Computability and computational complexity of the evolution of nonlinear dynamical systems. In Paola Bonizzoni, Vasco Brattka, and Benedikt Löwe, editors, *The Nature of Computation. Logic, Algorithms, Applications - 9th Conference on Computability in Europe, CiE 2013, Milan, Italy, July 1-5, 2013. Proceedings*, volume 7921 of *Lecture Notes in Computer Science*, pages 12–21. Springer, 2013. doi:10.1007/978-3-642-39053-1_2.

15 Olivier Bournez and Emmanuel Hainry. Elementarily computable functions over the real numbers and r-sub-recursive functions. *Theoretical Computer Science*, 348(2-3):130–147, 2005. doi:10.1016/j.tcs.2005.09.010.

16 Olivier Bournez and Amaury Pouly. A survey on analog models of computation. In Vasco Brattka and Peter Hertling, editors, *Handbook of computability and complexity in analysis*. Springer, 2021. doi:10.1007/978-3-030-59234-9_6.

17 Michael S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, 138(1):67–100, 6~February 1995. doi:10.1016/0304-3975(94)00147-B.

18 Manuel L. Campagnolo. *Computational complexity of real valued recursive functions and analog circuits*. Thèse de doctorat, IST, Universidade Técnica de Lisboa, 2001.

19 Manuel L. Campagnolo, Cristopher Moore, and José Félix Costa. An analog characterization of the subrecursive functions. In P. Kornerup, editor, *4th Conference on Real Numbers and Computers*, pages 91–109. Odense University Press, 2000.

20 Peter Clote and Evangelos Kranakis. *Boolean Functions and Computation Models*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2002. doi:10.1007/978-3-662-04943-3.

21 Alan Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam, 1962.

22 Riccardo Gozzi. *Analog Characterization of Complexity Classes*. PhD thesis, Instituto Superior Técnico, Lisbon, Portugal and University of Algarve, Faro, Portugal, 2022.

23 Daniel Silva Graça, Manuel Lameiras Campagnolo, and Jorge Buescu. Robust simulations of Turing machines with analytic maps and flows. In S. Barry Cooper, Benedikt Löwe, and Leen Torenvliet, editors, *New Computational Paradigms, First Conference on Computability in Europe, CiE 2005, Amsterdam, The Netherlands, June 8-12, 2005, Proceedings*, volume 3526 of *Lecture Notes in Computer Science*, pages 169–179. Springer-Verlag, 2005. doi:10.1007/11494645_21.

24 Daniel Silva Graça and José Félix Costa. Analog computers and recursive functions over the reals. *J. Complex.*, 19(5):644–664, 2003. doi:10.1016/S0885-064X(03)00034-7.

25 A. Grzegorczyk. Some classes of recursive functions, 1953.

26 Andrzej Grzegorczyk. Computable functionals. *Fundamenta Mathematicae*, 42:168–202, 1955. doi:10.4064/fm-42-1-168-202.

27 Ker-I Ko. *Complexity theory of real functions*, volume 3 of *Progress in theoretical computer science*. Birkhäuser, Boston, 1991. doi:10.1007/978-1-4684-6802-1.

28 Daniel Leivant and Jean-Yves Marion. Predicative functional recurrence and poly-space. In Michel Bidoit and Max Dauchet, editors, *TAPSOFT'97: Theory and Practice of Software Development, 7th International Joint Conference CAAP/FASE, Lille, France, April 14-18, 1997, Proceedings*, volume 1214 of *Lecture Notes in Computer Science*, pages 369–380. Springer, April 1997. doi:10.1007/BFb0030611.

29 Wolfgang Maass. Bounds for the computational power and learning complexity of analog neural nets. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 335–344. ACM, 1993. doi:10.1145/167088.167193.

30 Albert R. Meyer and Dennis M. Ritchie. The complexity of loop programs. In *Proceedings of the 1967 22nd National Conference (ACM '67)*, pages 465–469, New York, NY, USA, 1967. Association for Computing Machinery. doi:10.1145/800196.806014.

31 Jerzy Mycka and José Félix Costa. A new conceptual framework for analog computation. *Theor. Comput. Sci.*, 374(1-3):277–290, 2007. doi:10.1016/j.tcs.2007.01.005.

32 P. Odifreddi. *Classical Recursion Theory*, volume 2. Elsevier, 1999.

33 Amaury Pouly. *Continuous models of computation: from computability to complexity*. PhD thesis, Ecole Polytechnique and Unidersidade Do Algarve, defended on july 6, 2015 2015. https://pastel.archives-ouvertes.fr/tel-01223284, Prix de Thèse de l'Ecole Polyechnique 2016, Ackermann Award 2017.

34 Claude E. Shannon. Mathematical theory of the differential analyser. *Journal of Mathematics and Physics MIT*, 20:337–354, 1941. doi:10.1002/sapm1941201337.

35 John C. Shepherdson and Howard E. Sturgis. Computability of recursive functions. *Journal of the ACM*, 10(2):217–255, 1963. doi:10.1145/321160.321170.

36 Hava T. Siegelmann. *Neural networks and analog computation - beyond the Turing limit*. Progress in theoretical computer science. Birkhäuser, 1999.

37 Hava T. Siegelmann and Eduardo D. Sontag. Analog computation via neural networks. *Theoretical Computer Science*, 131(2):331–360, September 1994. doi:10.1016/0304-3975(94)90178-3.

38 Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, February 1995. doi:10.1006/jcss.1995.1013.

39 David B. Thompson. Subrecursiveness: Machine-independent notions of computability in restricted time and storage. *Mathematical Systems Theory*, 6(1):3–15, 1972. doi:10.1007/BF01706069.

40 Klaus Weihrauch. *Computable Analysis - An Introduction*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2000. doi:10.1007/978-3-642-56999-9.