Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning
Reasoning, Learning & Decision-Making Track

On the Expressivity of Recurrent Neural Cascades with Identity

Nadezda Alexandrovna Knorozova[1] and **Alessandro Ronca**[2]
[1]RelationalAI
[2]University of Oxford
nadezda.knorozova@relational.ai, alessandro.ronca@cs.ox.ac.uk

Abstract

Recurrent Neural Cascades (RNC) are the class of recurrent neural networks with no cyclic dependencies among recurrent neurons. Their subclass RNC+ with positive recurrent weights has been shown to be closely connected to the star-free regular languages, which are the expressivity of many well-established temporal logics. The existing expressivity results show that the regular languages captured by RNC+ are the star-free ones, and they leave open the possibility that RNC+ may capture languages beyond regular. We exclude this possibility for languages that include an *identity element*, i.e., an input that can occur an arbitrary number of times without affecting the output. Namely, in the presence of an identity element, we show that the languages captured by RNC+ are exactly the star-free regular languages. Identity elements are ubiquitous in temporal patterns, and hence our results apply to a large number of applications. The implications of our results go beyond expressivity. At their core, we establish a close structural correspondence between RNC+ and semiautomata cascades, showing that every neuron can be equivalently captured by a three-state semiautomaton. A notable consequence of this result is that RNC+ are no more succinct than cascades of three-state semiautomata.

1 Introduction

Recurrent Neural Cascades (RNCs) are a well-established formalism for learning temporal patterns. They are the subclass of recurrent neural networks where recurrent neurons are cascaded. Namely, they can be layed out into a sequence so that every neuron has access to the state of the preceding neurons as well as to the external input; and, at the same time, it has no dependency on the subsequent neurons.

RNCs admit several learning techniques. First, they admit general learning techniques for recurrent networks such as *backpropagation through time* (Werbos 1990), which learn the weights for a fixed architecture. Furthermore, the acyclic structure allows for constructive learning techniques such as *recurrent cascade correlation* (Fahlman 1990; Reed and Marks II 1999), which construct the cascade incrementally during training in addition to learning the weights.

RNCs have been successfully applied in many areas, including information diffusion in social networks (Wang et al. 2017), geological hazard prediction (Zhu et al. 2020), automated image annotation (Shin et al. 2016), intention recognition (Zhang et al. 2018), and optics (Xu et al. 2020).

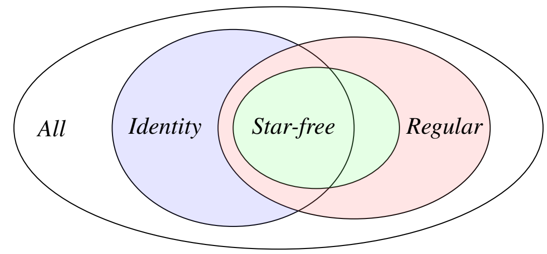

Figure 1: Relevant classes of languages. The label *'All'* denotes all formal languages, *'Identity'* denotes the languages with an identity element, *'Regular'* denotes the regular languages, and *'Star-free'* denotes the star-free regular languages.

Expressivity. We study the expressivity of RNCs in terms of *formal languages*, which provide a unifying framework where to describe the expressivity of all formalisms capturing temporal patterns. Early studies show there exist regular languages that are not captured by RNCs with monotone activation such as sigmoid and tanh (Giles et al. 1995; Kremer 1996). More recently the expressivity of RNCs has been studied in (Knorozova and Ronca 2024a). They show that the subclass RNC+ with positive recurrent weights captures all star-free regular languages, and it does not capture any other regular language. In terms of Figure 1, they show that the expressivity of RNC+ includes the green area, and it does not include the red area; leaving open any possibility for languages beyond regular. The correspondence with star-free regular languages makes RNC+ a strong candidate for learning temporal patterns. In fact, the star-free regular languages are a central class, corresponding to the expressivity of many well-known formalisms such as *star-free regular expressions* from where they take their name, *linear temporal logic* on finite traces (De Giacomo and Vardi 2013), *past temporal logic* (Manna and Pnueli 1991), *monadic first-order logic* on finite linear orders (McNaughton and Papert 1971), *group-free finite automata* (Ginzburg 1968), and *aperiodic finite automata* (Schützenberger 1965). However, there is still a possibility that RNC+ may capture patterns well-beyond the expressivity of such formalisms.

Our contribution. We extend the picture of the expressivity landscape of RNC+ by studying their capability to capture languages with an identity element. An *identity element* is an input that can occur an arbitrary number of times without affecting the output. We show that *a language with an identity element is recognised by* RNC+ *only if it is regular.* In other words, for any language beyond regular that has an identity element, we exclude the possibility that it is recognised by RNC+. In terms of Figure 1, we show that the blue area is not included in the expressivity of RNC+. Combined with the existing results, ours yields an exact characterisation of the expressivity of RNC+ in the presence of an identity element. Identity elements are ubiquitous, and hence the characterisation applies to a large number of relevant settings. Although we emphasise the results for languages, due to their importance, more generally our results apply to functions over strings.

Next we provide two examples of settings with identity. The first one is an example of a language defined by a temporal logic formula, and the second one is an example of an arithmetic function. More examples are given in Section 3.3.

Example 1 (Temporal Logics)**.** *Linear temporal logic allows for describing patterns over finite traces (De Giacomo and Vardi 2013). The formula $\varphi = \Diamond p$ holds whenever proposition p occurs at some point in the finite input trace. This defines a language L_φ over the alphabet $\Sigma = \{\emptyset, \{p\}\}$. The empty set is an identity element of L_φ.*

The language of Example 1 is star-free and thus it can be recognised by RNC+ by the results in (Knorozova and Ronca 2024a). Yet they have no implication for the following example. We show it cannot be implemented by RNC+.

Example 2 (Arithmetic)**.** *The function $F : \mathbb{Z}^+ \to \mathbb{Z}$ returns the sum of the input integers as $F(z_1 \dots z_\ell) = z_1 + \cdots + z_\ell$. The number 0 is an identity element of F.*

Technically, at the core of our results we show a close structural correspondence between RNC+ and cascades of finite semiautomata. One can start from a given RNC+ and obtain an equivalent cascade of semiautomata by replacing each recurrent neuron with a three-state semiautomaton. A cascade of three-state semiautomata is itself a finite-state semiautomaton. This implies our expressivity results mentioned above, as well as succinctness results. For instance, any language that requires a cascade of n three-state semiautomata cannot be captured by RNC+ with fewer than n recurrent neurons. In this sense, RNC+ is no more succinct than semiautomata cascades. In turn, the former implies that RNC+ with n recurrent neurons cannot recognise a language that requires an automaton with more than 3^n states.

Proofs of all our results are included, with some deferred to the extended version (Knorozova and Ronca 2024b).

2 Preliminaries

We denote the natural numbers by $\mathbb{N}$, the real numbers by $\mathbb{R}$, and the non-negative real numbers by $\mathbb{R}_+$. For $n \in \mathbb{N}$, we write $[n]$ for the set $\{1, 2, \dots, n\} \subseteq \mathbb{N}$. We write an infinite sequence $(a_k, a_{k+1}, \dots)$ as $(a_t)_{t \geq k}$. Given a factored set $Z \subseteq Z_1 \times \cdots \times Z_n$ and an index $i \in [n]$, we define the projection of Z on its first i components as

$$Z_{[i]} = \{\langle z_1, \dots, z_i \rangle \mid \exists z_{i+1}, \dots, z_n. \langle z_1, \dots, z_n \rangle \in Z\}.$$

When we apply a function $f : X \to Y$ to a subset $Z \subseteq X$ of its inputs, the result is the set $f(Z) = \{f(x) \mid x \in Z\}$.

Equivalence relations. An *equivalence relation* $\sim$ over a set X is a binary relation that is reflexive, symmetric, and transitive. The *equivalence class* of $x \in X$, written as $[\![x]\!]$, is the set of all elements in X that are equivalent to x. The set of all equivalence classes is a partition of X, and it is written as $X/{\sim}$. Sometimes we name an equivalence relation as $\sim_a$, and write the corresponding equivalence classes as $[\![x]\!]_a$.

Metric spaces and continuous functions. A *metric space* is a set X equipped with a function $d_X : X \times X \to \mathbb{R}$ called a *metric* which satisfies the properties: (i) $d_X(x, x) = 0$, (ii) $d_X(x, y) \neq 0$ when $x \neq y$, (iii) $d_X(x, y) = d_X(y, x)$, (iv) $d_X(x, z) \leq d_X(x, y) + d_X(y, z)$. It is *discrete* if the metric satisfies $d_X(x, y) = 1$ for $x \neq y$ and $d_X(x, y) = 0$ for $x = y$. Every set can be made a discrete space. For X and Y metric spaces, a function $f : X \to Y$ is *continuous at a point* $c \in X$ if, for every positive real number $\epsilon > 0$, there exists a positive real number $\delta > 0$ such that every $x \in X$ satisfying $d_X(x, c) < \delta$ also satisfies $d_Y(f(x), f(c)) < \epsilon$. Equivalently, function f is *continuous at a point* $c \in X$ if, for every sequence $(x_t)_{t \geq 0}$ of elements of X with limit c, it holds that the limit of the sequence $(f(x_t))_{t \geq 0}$ is $f(c)$. Function f is *continuous* if it is so at every point $c \in X$.

2.1 Dynamical Systems

Dynamical systems provide us with a formalism where to cast both recurrent neural cascades and automata. A *dynamical system* S is a tuple

$$S = \langle U, X, f, x^{\text{init}}, Y, h \rangle,$$

where U is a set of elements called *inputs*, X is a set of elements called *states*, $f : X \times U \to X$ is called *dynamics function*, $x^{\text{init}} \in X$ is called *initial state*, Y is a set of elements called *outputs*, and $h : X \to Y$ is called *output function*. Sets U, X, Y are equipped with a metric. System S is *continuous* if functions f and h are continuous.

At every time point $t = 1, 2, \dots$, the system receives an input $u_t \in U$. The state x_t and output y_t of the system at time t are defined as follows. At time $t = 0$, before receiving any input, the system is in state $x_0 = x^{\text{init}}$ and the output is $y_0 = h(x^{\text{init}})$. Then, the state x_t is determined by the previous state x_{t-1} and the current input u_t, and consequently the output y_t is determined by x_t, as

$$x_t = f(x_{t-1}, u_t), \qquad y_t = h(x_t).$$

The *dynamics* of S are the tuple $D = \langle U, X, f \rangle$. Dynamics D are *continuous* if f is continuous. *Subdynamics* of D are any tuple $\langle U, X', f \rangle$ such that $X' \subseteq X$ and $f(X', U) \subseteq X'$. The function *implemented* by system S is the function that maps every input sequence $u_1, \dots, u_\ell$ to the output y_ℓ. We write $S(u_1, \dots, u_\ell) = y_\ell$. Such function is also defined on the empty input sequence, in which case it returns y_0. Two systems are *equivalent* if they implement the same function.

Homomorphic representation. The notion of homomorphic representation allows for comparing systems by relating their dynamics. We follow (Knorozova and Ronca

2024a). Consider two system dynamics $D_1 = \langle U, X_1, f_1 \rangle$ and $D_2 = \langle U, X_2, f_2 \rangle$. A *homomorphism* from D_1 to D_2 is a continuous surjective function $\psi : X_1 \to X_2$ satisfying

$$\psi\big(f_1(x, u)\big) = f_2\big(\psi(x), u\big)$$

for every state $x \in X_1$ and every input $u \in U$. Dynamics D_1 *homomorphically represents* D_2 if D_1 has subdynamics D_1' such that there is a homomorphism from D_1' to D_2.

First, homomorphic representation has the following implication on the existence of an equivalent system.

Proposition 1. *If dynamics D_1 homomorphically represent the dynamics of a system S_2, then there is a system S_1 with dynamics D_1 that is equivalent to S_2.*

Second, equivalence of two systems implies homomorphic representation, but only under certain conditions which include canonicity—a notion that we introduced next. A state x of a system S is *reachable* if there is an input sequence $u_1, \dots, u_t$ such that the system is in state x at time t. A system is *connected* if every state is reachable. Given a system S and one of its states x, the system S^x is the system obtained by setting x to be the initial state. Two states x and x' of S are equivalent if the systems S^x and $S^{x'}$ are equivalent. A system is in *reduced form* if it has no distinct states which are equivalent. A system is *canonical* if it is connected and in reduced form.

Proposition 2. *If a continuous system S_1 is equivalent to a canonical system S_2 with a discrete output, then the dynamics of S_1 homomorphically represent the dynamics of S_2.*

2.2 Cascade Architecture

A *cascade* C is a form of dynamics $\langle U, X, f \rangle$ with a factored set of states $X = X_1 \times \cdots \times X_n$ and dynamics function of the form

$$f(\langle x_1, \dots, x_n \rangle, u) = \langle f_1(x_1, u_1), \dots, f_n(x_n, u_n) \rangle,$$
$$\text{where } u_i = \langle u, x_1, \dots, x_{i-1} \rangle.$$

Function f_i determines the i-th element of the next state based on the input u and the first $i-1$ elements of the current state. It is convenient to also introduce the function that returns the first i elements

$$\bar{f}_i(\langle x_1, \dots, x_i \rangle, u) = \langle f_1(x_1, u_1), \dots, f_i(x_i, u_i) \rangle.$$

Adopting a modular view, we can see cascade C as consisting of n dynamics $D_1, \dots, D_n$ where

$$D_i = \langle U \times X_{[i-1]}, X_i, f_i \rangle.$$

We call every D_i a *component* of the cascade, and we write $C = D_1 \ltimes \cdots \ltimes D_n$. Every component has access to the state of the preceding components but not to the state of the subsequent components, avoiding cycling dependencies.

2.3 Recurrent Neural Cascades

A *core recurrent tanh neuron* is a triple $N = \langle V, X, f \rangle$ where $V \subseteq \mathbb{R}$ is the input domain, $X \subseteq \mathbb{R}$ are the states, and f is the function

$$f(x, v) = \tanh(w \cdot x + v),$$

with $w \in \mathbb{R}$ called *recurrent weight*. A *recurrent tanh neuron* is the composition of a core recurrent tanh neuron N with an *input function* $\beta : U \subseteq \mathbb{R}^a \to V$ that can be implemented by a feedforward neural network. Namely, it is a triple $\langle U, X, f_\beta \rangle$ where $f_\beta(x, u) = f(x, \beta(u))$. A recurrent tanh neuron is a form of dynamics, so the notions for dynamical systems apply. We will mostly omit the term 'recurrent tanh' as it is the only kind of neuron we consider explicitly.

A *Recurrent Neural Cascade (RNC)* is a dynamical system whose dynamics are a cascade of recurrent tanh neurons and whose output function can be implemented by a feedforward neural network. An RNC+ is an RNC where all recurrent weights are positive.

2.4 Automata

Automata are dynamical systems, but the terminology employed is different. The input and output domains are called *alphabets*, and their elements are called *letters*. Input and output sequences are seen as *strings*, where a string $\sigma_1 \dots \sigma_\ell$ is simply a concatenation of letters. The set of all strings over an alphabet Σ is written as Σ^*. An *automaton* is a tuple $A = \langle \Sigma, Q, \delta, q^{\text{init}}, \Gamma, \theta \rangle$ where Σ is called *input alphabet* (rather than input domain), Q is the set of states, $\delta : Q \times \Sigma \to Q$ is called *transition function* (rather than dynamics function), $q^{\text{init}} \in Q$ is the initial state, Γ is called *output alphabet* (rather than output domain), and $\theta : Q \to \Gamma$ is the output function. The tuple $D = \langle \Sigma, Q, \delta \rangle$ is called a *semiautomaton*, rather than dynamics. For every $\sigma \in \Sigma$, the function $\delta_\sigma(q) = \delta(q, \sigma)$ is called a *transformation* of the semiautomaton D; it is an *identity transformation* if $\delta_\sigma(q) = q$ for every $q \in Q$. States and alphabets of an automaton are allowed to be infinite. If an automaton has a finite number of states we say it is a *finite-state automaton*. Given a semiautomaton $\langle \Pi, Q, \delta \rangle$ and a function $\phi : \Sigma \to \Pi$, their composition is the semiautomaton $\langle \Sigma, Q, \delta_\phi \rangle$ whose transition function is $\delta_\phi(q, \sigma) = \delta(q, \phi(\sigma))$.

2.5 Classes of Languages and Functions

The set of all strings over an alphabet Σ is denoted by Σ^*. A *language* L over a finite Σ is a subset of Σ^*. Language L can also be seen as the indicator function $f_L : \Sigma^* \to \{0, 1\}$ where $f_L(x) = 1$ iff $x \in L$. An *automaton acceptor* is an automaton whose output alphabet is $\{0, 1\}$. An automaton acceptor *recognises* L if it implements f_L. The *regular languages* are the ones that can be expressed by regular expressions, and they coincide with the languages that can be recognised by finite-state automaton acceptors (Kleene 1956). The *star-free regular languages* are the ones that can be expressed by star-free regular expressions, and they coincide with the aperiodic regular languages also known as noncounting regular languages, cf. (Ginzburg 1968). A language L is *aperiodic* if there exists a non-negative integer n such that, for all strings $x, y, z \in \Sigma^*$, we have $xy^n z \in L$ iff $xy^{n+1} z \in L$. The characterisations for languages generalise to functions $f : \Sigma^* \to \Gamma$ in the following way. A function is *regular* if it can be implemented by a finite-state automaton. A function F is *aperiodic* if there exists a non-negative integer n such that, for all strings $x, y, z \in \Sigma^*$, the equality $F(xy^n z) = F(xy^{n+1} z)$ holds.

3 Expressivity of RNC+

In this section we present our results. We begin by introducing the setting in Section 3.1, and then briefly reporting the existing expressivity results in Section 3.2. The core of our contribution is in Sections 3.3, 3.4, and 3.5. In particular, Section 3.3 introduces the notion of identity element for languages and functions, discussing several examples; Section 3.4 presents our core technical results; and Section 3.5 presents our expressivity results.

3.1 Setting

Our goal is to establish expressivity results for RNC+. We consider throughout the section an *input alphabet* Σ and an *output alphabet* Γ. Then the goal is to establish which functions from Σ^* to Γ can be implemented by RNC+, which however operate on real-valued input domain $U \subseteq \mathbb{R}^a$ and output domain $Y \subseteq \mathbb{R}^b$. To close the gap while staying general, we avoid identifying U with Σ and Y with Γ. Instead, we introduce mappings between such sets, that can be regarded as symbol groundings.

Definition 1. *Given a domain $Z \subseteq \mathbb{R}^n$ and an alphabet Λ, a* symbol grounding *from Z to Λ is a continuous surjective function $\lambda : Z \to \Lambda$.*

Symbol groundings can be seen as connecting the subymbolic level $Z \subseteq \mathbb{R}^n$ to the symbolic level Λ. For an element z at the subsymbolic level, the letter $\lambda(z)$ is its meaning at the symbolic level. Assuming that a symbol grounding λ is surjective means that every letter corresponds to at least one element $z \in Z$. The assumption is w.l.o.g. because we can remove the letters that do not represent any element of the subsymbolic level.

We fix an *input symbol grounding* $\lambda_\Sigma : U \to \Sigma$ and an *output symbol grounding* $\lambda_\Gamma : Y \to \Gamma$. Then we say that an RNC+ N implements a function $F : \Sigma^* \to \Gamma$ if, for every input string $u_1 \dots u_t \in U^*$, the following equality holds.

$$\lambda_\Gamma\big(N(u_1 \dots u_t)\big) = F\big(\lambda_\Sigma(u_1) \dots \lambda_\Sigma(u_t)\big)$$

Specifically for languages, we have Σ finite and $\Gamma = \{0, 1\}$, and we say that an RNC+ recognises L if it implements its indicator function f_L. Note that symbol groundings are w.l.o.g. since one can choose them to be identity. In this case, implementing a function under symbol groundings coincides with the default notion of implementing a function.

3.2 Existing Expressivity Results

We report the existing expressivity results for RNC+.

Theorem 1 (Knorozova and Ronca, 2024)**.** *The regular languages recognised by* RNC+ *are the star-free regular languages. The regular functions over finite alphabets implemented by* RNC+ *are the aperiodic regular functions.*

Note, in particular, that the results have no implication for languages and functions that are not regular.

3.3 Languages and Functions with Identity

We introduce the notion of identity element for languages and functions, and we discuss several examples.

Definition 2. *A letter $e \in \Sigma$ is an* identity element *for a language L over Σ if, for every pair of strings $x, y \in \Sigma^*$, it holds that $xy \in L$ if and only if $xey \in L$.*

The above definition generalises to functions as follows.

Definition 3. *A letter $e \in \Sigma$ is an* identity element *for a function $F : \Sigma^* \to \Gamma$ if, for every pair of strings $x, y \in \Sigma^*$, it holds that $F(xy) = F(xey)$.*

Note that e is an identity element for a language L iff it is an identity element for its indicator function f_L. Next we present examples of languages and functions from different application domains.

Example 3 (Reinforcement Learning)**.** *In reinforcement learning, agents are rewarded according to the history of past events. Consider an agent that performs navigational tasks in a grid. At each step, the agent moves into one direction by one cell or stays in the same cell, which is communicated to us using the propositions $\Sigma = \{stayed, left, right, up, down\}$. We know the initial position (x_0, y_0), and we reward the agent when it visits a goal position (x_g, y_g). This amounts to a language over Σ, for which the proposition $stayed$ is an identity element.*

When the grid of the above example is finite, the resulting language can be recognised by RNC+ as a consequence of Theorem 1 since the language is star-free regular.

Example 4 (Temporal Logic)**.** *The temporal logic Past LTL allows for describing patterns over traces using past operators (Manna and Pnueli 1991). The Past LTL formula $\varphi = p \mathcal{S} q$ holds whenever proposition p has always occurred since the latest occurrence of q. This defines a language L_φ over the alphabet $\Sigma = \{\emptyset, \{p\}, \{q\}, \{p, q\}\}$. The letter $\sigma_p = \{p\}$ is an identity element for L_φ.*

The language of the above example can be recognised by RNC+ according to Theorem 1 since it is star-free regular.

Example 5 (Arithmetic Functions)**.** *The following ones are examples of arithmetic functions with an identity element.*

- *F_1 takes a list of natural numbers and returns their product, as $F_1(n_1 \dots n_\ell) = n_1 \times \cdots \times n_\ell$.*
- *F_2 takes a list of reals and returns the sign of their sum, as $F_2(r_1 \dots r_\ell) = \mathrm{sign}(r_1 + \cdots + r_\ell)$.*
- *F_3 takes a list of bits $\{0, 1\}$ and indicates whether they sum to 16, as $F_3(n_1 \dots n_\ell) = [n_1 + \cdots + n_\ell = 16]$.*
- *F_4 takes a list of integers from $[0, 6]$ and returns their sum modulo 7, as $F_4(n_1 \dots n_\ell) = n_1 + \cdots + n_\ell \mod 7$.*
- *F_5 takes a list of increments $\{-1, 0, +1\}$ and returns the sign of their sum, as $F_5(z_1 \dots z_\ell) = \mathrm{sign}(z_1 + \cdots + z_\ell)$.*

The identity element of F_1 is 1, the identity element of F_2, F_3, F_4, and F_5 is 0.

Theorem 1 implies that function F_3 of the above example can be implemented by RNC+ since it is aperiodic regular, and also that function F_4 cannot be implemented by RNC+ since it is regular but not aperiodic.

3.4 Our Core Results

This section presents our core technical results. First, we show that identity elements imply identity transformations.

Proposition 3. *A canonical automaton implements a function with an identity element only if it has an identity transformation.*

Proof. Let F be a function from Σ^* to Γ having an identity element $e \in \Sigma$. Let $A = \langle \Sigma, Q, \delta \rangle$ be a canonical automaton that implemements F. Let us consider the transformation $\delta_e(q) = \delta(q, e)$ of A. We show that δ_e is an identity transformation. Let $q \in Q$, and let $q' = \delta_e(q)$. It suffices to show $q = q'$. Since A is canonical and hence connected, there exists a string s that leads to q from the initial state. Then, the string se leads from the initial state to q'. For every string s', we have $A^q(s') = A(ss') = F(ss')$ and similarly we have $A^{q'}(s') = A(ses') = F(ses')$. We have $F(ss') = F(ses')$ since e is an identity element for F, and hence the equalities above imply $A^q(s') = A^{q'}(s')$. Then the required equality follows immediately by canonicity of A. □

Technically, the following lemma is our core result.

Lemma 1. *Let D be the dynamics of an* RNC+ *with n components, and let A_T be a semiautomaton with an identity transformation. Let A_Σ be the composition of A_T with the input symbol grounding λ_Σ. If D homomorphically represents A_Σ, then A_T is homomorphically represented by a cascade of n three-state semiautomata.*

Proof. See Section 4. □

Equipped with the lemma above, we can now characterise the functions that an RNC+ can implement.

Theorem 2. *Let F be a function from Σ^* to Γ that has an identity element, with Γ discrete. If F is implemented by an* RNC+ *with n neurons, then there exists an automaton that implements F and whose semiautomaton is a cascade of n three-state semiautomata.*

Proof. Let N be an RNC+ with n neurons that implements F. Furthermore, let A be a canonical automaton that implements F, which always exists. By Proposition 3, we have that A has an identity transformation. Since N is equivalent to A, by Proposition 2, we have that the dynamics of N homomorphically represent the semiautomaton of A. Then, by Lemma 1, it follows that the semiautomaton of A is homomorphically represented by a cascade C of n three-state semiautomata. By Proposition 1, there is an automaton A_C with semiautomaton C that is equivalent to A, and hence it implements F. □

The above theorem can be interpreted as providing a lower bound on the succinctness of RNC+. Namely, if a function requires at least n components to be implemented by a cascade of three-state semiautomata, then it necessarily requires an RNC+ with at least n neurons.

In particular, the theorem immediately implies a finite bound on the number of states required to implement any function that can be implemented by an RNC+.

Corollary 1. *Let F be a function from Σ^* to Γ that has an identity element, with Γ discrete. If F is implemented by an* RNC+ *with n neurons, then there exists an automaton with at most 3^n states that implements F.*

The corollary can be interpreted as providing a lower bound on the succinctness of RNC+. Namely, an RNC+ with n components cannot implement a function that requires more than 3^n states.

Remark 1. *Theorem 2 and Corollary 1 apply to languages seamlessly, as they apply to their indicator function.*

3.5 Our Expressivity Results

In this section we state our expressivity results for functions, and hence languages, with an identity element.

Theorem 3. *The functions with an identity element and a discrete codomain implemented by* RNC+ *are regular.*

Proof. Let us consider a function F with an identity element and a discrete codomain, and let N be an RNC+ that implements F. By Theorem 2, there exists an automaton A that implements F and whose semiautomaton is a cascade of three-state semiautomata. In particular, A is finite-state and hence F is regular. □

We combine our results for functions with the existing ones to obtain an exact characterisation of the functions over finite alphabets recognised by RNC+ in the presence of an identity element.

Theorem 4. *The functions over finite alphabets having an identity element that can be implemented by* RNC+ *are aperiodic regular.*

Proof. Consider a function F over finite alphabets with an identity element implemented by RNC+. We have that F is regular by Theorem 3, noting that every finite alphabet is discrete. Then, F is aperiodic regular by Theorem 1. □

Having established the results for functions, we now derive the result for languages, considering that their indicator function is a function over finite alphabets.

Theorem 5. *The languages having an identity element that can be recognised by* RNC+ *are star-free regular.*

Proof. Let L be a language with an identity element, and let f_L be its indicator function. An RNC+ recognises L if it implements f_L. By Theorem 3, it follows that f_L is regular. We conclude that L is regular, and hence star-free regular by Theorem 1. □

Theorems 3–5 allow us to draw the missing conclusions for the languages and functions of the examples from the previous sections. First, Theorem 3 implies that function F of Example 2 and functions F_1 and F_2 of Example 5 cannot be implemented by RNC+ since they are not regular. Second, Theorem 4 implies that function F_5 of Example 5 cannot be implemented by RNC+ since it is not regular. Third, Theorem 5 implies that the language of Example 3 cannot be recognised by RNC+ when the grid is infinite, since the language is not regular in this case.

4 Proof of Lemma 1

In this section we prove Lemma 1. We first introduce the context in Section 4.1 below. Ultimately we will construct the required cascade in Section 4.5. To do that, we establish several intermediate results in Sections 4.2, 4.3, and 4.4.

4.1 Context

We introduce the context of the proof. Let $D = \langle X, U, f\rangle$ be the dynamics of an RNC+. We have $D = N_1 \ltimes \cdots \ltimes N_n$ where $N_i = \langle X_i, U_i, f_i\rangle$ is a recurrent tanh neuron, with dynamics function

$$f_i(x_i, u_i) = \tanh(w_i \cdot x_i + \beta_i(u_i)),$$

with input $u_i = \langle u, x_1, \dots, x_{i-1}\rangle$ and weight $w_i \in \mathbb{R}_+$. Let $A_T = \langle Q_T, \Sigma, \delta_T\rangle$ be a semiautomaton with an identity transformation induced by a letter $e \in \Sigma$. Let A_Σ be the semiautomaton resulting from the composition of A_T with the input symbol grounding λ_Σ. Namely, $A_\Sigma = \langle U, Q_T, \delta_\Sigma\rangle$ with $\delta_\Sigma(q, u) = \delta_T(q, \lambda_\Sigma(u))$. Let $u_e \in U$ be an input such that $e = \lambda_\Sigma(u_e)$, which exists since λ_Σ is surjective.

The assumption is that A_Σ is homomorphically represented by D. Thus, there exists a homomorphism ψ from some subdynamics $D' = \langle X', U, f\rangle$ of D to A_Σ.

4.2 Convergence results

We show that the sequence of states of any RNC+ upon receiving a repeated input is convergent. In particular, it converges to a fixpoint of the dynamics function of the RNC+. We introduce notation to refer to such a sequence of states.

Definition 4. *Let $u \in U$, let $\langle x_1, \dots, x_n\rangle \in X$. For every $i \in [n]$, we define the sequence $(x_{i,t})_{t\geq 0}$ as*

$$x_{i,0} = x_i,$$
$$x_{i,t} = f_i(x_{i,t-1}, \langle x_{1,t-1}, \dots, x_{i-1,t-1}, u\rangle) \quad \textit{for } t \geq 1,$$

and we refer to it by $\mathcal{S}_i(u, x_1, \dots, x_i)$. For every $i \in [n]$ and every index $t \geq 0$, we define

$$\mathcal{S}_i^t(u, x_1, \dots, x_i) = x_{i,t},$$
$$\bar{\mathcal{S}}_i^t(u, x_1, \dots, x_i) = \langle x_{1,t}, \dots, x_{i,t}\rangle.$$

We show the sequence of states to be convergent, adapting an argument from (Knorozova and Ronca 2024a).

Proposition 4. *Let $i \in [n]$, let $u \in U$, and let $\mathbf{x} \in X_{[i]}$. The sequence $\mathcal{S}_i(u, \mathbf{x})$ is convergent.*

In light of the above proposition, we introduce notation to refer to the limit of the converging sequence of states.

Definition 5. *Let $i \in [n]$, let $u \in U$, and let $\mathbf{x} \in X_{[i]}$. We define $\mathcal{S}_i^*(u, \mathbf{x})$ as the limit of the sequence $\mathcal{S}_i(u, \mathbf{x})$. Furthermore, we define $\bar{\mathcal{S}}_i^*(u, \mathbf{x}) = \langle x_{1,*}, \dots, x_{i,*}\rangle$ where $x_{j,*} = \mathcal{S}_j^*(u, x_1, \dots, x_j)$ for every $j \in [i]$.*

Next we show that the sequence converges to a fixpoint.

Proposition 5. *Let $i \in [n]$, let $u \in U$, and let $\mathbf{x} \in X_{[i]}$. The sequence $\mathcal{S}_i(u, \mathbf{x})$ converges to a fixpoint of the function $h_{i,v}(x) = \tanh(w_i \cdot x + v)$ for $v = \beta_1(u)$ when $i = 1$ and $v = \beta_i\big(u, \bar{\mathcal{S}}_{i-1}^*(u, \mathbf{x})\big)$ when $i \geq 2$.*

Proof. Let $(x_{i,t})_{t\geq 0}$ be the sequence $\mathcal{S}_i(u, \mathbf{x})$, and let $\bar{\mathcal{S}}_i^*(u, \mathbf{x}) = \langle x_{1,*}, \dots, x_{i,*}\rangle$. For every $j \in [i]$, we have

$$\lim_{t\to\infty} x_{j,t} = x_{j,*}.$$

By continuity of f_i, we have

$$\begin{aligned}&\lim_{t\to\infty} f_i(x_{i,t}, \langle u, x_{1,t}, \dots, x_{i-1,t}\rangle)\\ &= f_i(x_{i,*}, \langle u, x_{1,*}, \dots, x_{i-1,*}\rangle)\\ &= h_{i,v}(x_{i,*}).\end{aligned}$$

By the definition of $x_{i,t+1}$, we have

$$\lim_{t\to\infty} f_i(x_{i,t}, \langle u, x_{1,t}, \dots, x_{i-1,t}\rangle) = \lim_{t\to\infty} x_{i,t+1} = x_{i,*}.$$

Thus $h_{i,v}(x_{i,*}) = x_{i,*}$, hence $x_{i,*}$ is a fixpoint of $h_{i,v}$. □

4.3 Equivalence classes

Based on the convergence results, we introduce equivalence relations which describe the necessary behaviour of the considered homomorphism ψ. Here the focus is on the relevant subdynamics D' of the RNC+. Let us recall that X' is the set of states of D', it is a factored set, and $X'_{[i]}$ denotes its projection on the first i components. Elements of $X'_{[i]}$ are states of the prefix $N_1 \ltimes \cdots \ltimes N_i$ of the RNC+ dynamics. We introduce an equivalence relation on $X'_{[i]}$ based on where states converge when the identity input u_e is repeatedly applied.

Definition 6. *For every $i \in [n]$, we define the equivalence relation $\sim_e$ on $X'_{[i]}$ as the smallest equivalence relation such that, for every $\mathbf{x}, \mathbf{y} \in X_{[i]}$, the equivalence $\mathbf{x} \sim_e \mathbf{y}$ holds whenever $\bar{\mathcal{S}}_i^*(u_e, \mathbf{x}) = \bar{\mathcal{S}}_i^*(u_e, \mathbf{y})$.*

Next we coarsen the above equivalence relation by making equivalent the successor states of equivalent states.

Definition 7. *For every $i \in [n]$, we define the equivalence relation $\sim$ on $X'_{[i]}$ as the smallest equivalence relation such that, for every $\mathbf{x}, \mathbf{y} \in X'_{[i]}$, the following implications hold:*

- $\mathbf{x} \sim_e \mathbf{y}$ *implies* $\mathbf{x} \sim \mathbf{y}$;
- $\mathbf{x} \sim \mathbf{y}$ *implies* $\bar{f}_i(\mathbf{x}, u) \sim \bar{f}_i(\mathbf{y}, v)$ *for every* $u, v \in U$ *with* $\lambda_\Sigma(u) = \lambda_\Sigma(v)$.

In the next proposition we show that, when input u_e is iterated, the homomorphism maps all states of the resulting sequence to the same state of the target semiautomaton.

Proposition 6. *For every $\mathbf{x} \in X'$ and every index $t \geq 0$, it holds that $\psi(\bar{\mathcal{S}}_n^t(u_e, \mathbf{x})) = \psi(\bar{\mathcal{S}}_n^*(u_e, \mathbf{x})) = \psi(\mathbf{x})$.*

Proof. Let $\mathbf{x} \in X'$. For every $t \geq 1$, by definition we have $\bar{\mathcal{S}}_n^t(u_e, \mathbf{x}) = f(\bar{\mathcal{S}}_n^{t-1}(u_e, \mathbf{x}), u_e)$. Then, by the definition of homomorphism, and since $\lambda(u_e) = e$ induces an identity transformation in A_T, the following holds for every $t \geq 1$,

$$\begin{aligned}\psi(\bar{\mathcal{S}}_n^t(u_e, \mathbf{x})) &= \psi(f(\bar{\mathcal{S}}_n^{t-1}(u_e, \mathbf{x}), u_e))\\ &= \delta_T(\psi(\bar{\mathcal{S}}^{t-1}(u_e, \mathbf{x}), e)) = \psi(\bar{\mathcal{S}}^{t-1}(u_e, \mathbf{x})).\end{aligned}$$

and hence $\psi(\bar{\mathcal{S}}_n^t(u_e, \mathbf{x})) = \psi(\bar{\mathcal{S}}_n^0(u_e, \mathbf{x})) = \psi(\mathbf{x})$. Then, by continuity of ψ,

$$\begin{aligned}\psi(\bar{\mathcal{S}}_n^*(u_e, \mathbf{x})) &= \psi\big(\lim_{t\to\infty} \bar{\mathcal{S}}_n^t(u_e, \mathbf{x})\big)\\ &= \lim_{t\to\infty} \psi(\bar{\mathcal{S}}_n^t(u_e, \mathbf{x})) = \lim_{t\to\infty} \psi(\mathbf{x}) = \psi(\mathbf{x}).\end{aligned}$$

This concludes the proof of the proposition. □

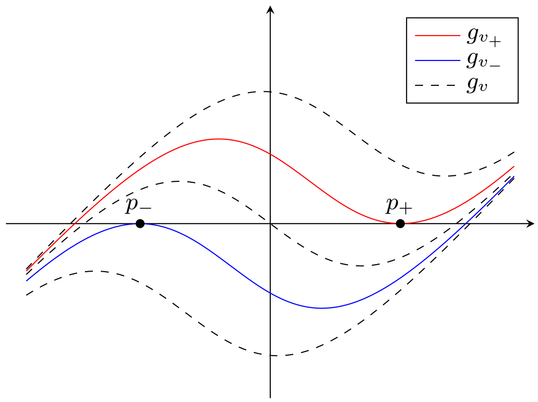

Figure 2: Function g_v for different values of v.

Finally we develop an inductive argument to show that, starting from two states that are treated equally by the homomorphism, their successors will also be treated equally. And thus the homomorphism is overall invariant under the coarser of our equivalence relations.

Proposition 7. *For every $\mathbf{x}, \mathbf{y} \in X'$, it holds that $\mathbf{x} \sim \mathbf{y}$ implies $\psi(\mathbf{x}) = \psi(\mathbf{y})$.*

Proof. Since $\mathbf{x} \sim \mathbf{y}$, there exist states $\mathbf{x}_0, \mathbf{y}_0 \in X'$ with $\mathbf{x}_0 \sim_e \mathbf{y}_0$, and two possibly-empty sequences of inputs $u_1, \dots, u_t$ and $v_1, \dots, v_t$ such that (i) $\lambda_\Sigma(u_k) = \lambda_\Sigma(v_k)$ for every $k \in [t]$, and (ii) letting $\mathbf{x}_k = f(\mathbf{x}_{k-1}, u_k)$ and $\mathbf{y}_k = f(\mathbf{y}_{k-1}, v_k)$ for every $k \in [t]$, we have $\mathbf{x}_t = \mathbf{x}$ and $\mathbf{y}_t = \mathbf{y}$. We prove the proposition by induction on t.

In the base case $t = 0$, hence $\mathbf{x}_0 = \mathbf{x}$ and $\mathbf{y}_0 = \mathbf{y}$, and hence $\mathbf{x} \sim_e \mathbf{y}$. Thus $\bar{\mathcal{S}}^*_n(u_e, \mathbf{x}) = \bar{\mathcal{S}}^*_n(u_e, \mathbf{y})$, and hence by Proposition 6 we have $\psi(\mathbf{x}) = \psi(\mathbf{y})$ as required.

In the inductive case, we have $t \geq 1$ and we assume $\psi(\mathbf{x}_{t-1}) = \psi(\mathbf{y}_{t-1})$. By the definition of homomorphism,

$$\psi(\mathbf{x}_t) = \psi(f(\mathbf{x}_{t-1}, u_t)) = \delta_T\big(\psi(\mathbf{x}_{t-1}), \lambda_\Sigma(u_t)\big),$$
$$\psi(\mathbf{y}_t) = \psi(f(\mathbf{y}_{t-1}, v_t)) = \delta_T\big(\psi(\mathbf{y}_{t-1}), \lambda_\Sigma(v_t)\big).$$

Since $\psi(\mathbf{x}_{t-1}) = \psi(\mathbf{y}_{t-1})$ and $\lambda_\Sigma(u_t) = \lambda_\Sigma(v_t)$, we conclude that $\psi(\mathbf{x}_t) = \psi(\mathbf{y}_t)$, as required. ☐

4.4 Analysis of tanh dynamics

We carry out an analysis of the dynamics function of a recurrent tanh neuron, with the goal of identifying its fixpoints and in particular the way they are positioned. Let $w \in \mathbb{R}_+$, let $v \in \mathbb{R}$, and let us consider the functions

$$h_v(x) = \tanh(w \cdot x + v), \qquad g_v(x) = x - h_v(x).$$

Function h_v is the dynamics function of a recurrent neuron for a fixed input, and its fixpoints coincide with the zeroes of g_v. In fact, $h_v(x) = x$ iff $g_v(x) = 0$. Hence, we analyse the zeroes of g_v in place of the fixpoints of h_v.

Proposition 8. *If $w \in [0, 1]$, function g_v has only one zero.*

In the rest we consider the case of $w > 1$. The graph of g_v for different values of v is shown in Figure 2. As it can be seen from the graph, going from left to right, the function is increasing, then decreasing, and then increasing again. In particular, it has two stationary points.

Proposition 9. *The following properties hold:*

- *$g_v(x)$ goes to $-\infty$ when $x \to -\infty$,*
- *$g_v(x)$ goes to $+\infty$ when $x \to +\infty$,*
- *g_v has exactly two stationary points $p^v_- < p^v_+$,*
- *g_v is strictly increasing in $(-\infty, p^v_-) \cup (p^v_+, +\infty)$,*
- *g_v is strictly decreasing in the interval (p^v_-, p^v_+).*

In particular, p^v_- is a local maximum of g_v, and p^v_+ is a local minimum of g_v. Furthermore, the derivative of g_v is bounded as $g'_v(x) \in [0, 1]$ for every $x \in [-1, p^v_-] \cup [p^v_+, +1]$.

Different values of v determine different diagonal translations of the same curve, as it can be observed from Figure 2. They also determine different horizontal translations of the derivative g'_v, which allows us to determine how stationary points are translated for different values of v.

Proposition 10. *Let $u, v \in \mathbb{R}$, and let $d = (u - v)/w$. It holds that $g_u(x) = g_v(x + d) - d$ and $g'_u(x) = g'_v(x + d)$. Furthermore, $p^u_+ = p^v_+ - d$ and $p^u_- = p^v_- - d$.*

Thus, depending on v, the function g_v crosses the x axis in one, two, or three points. Of particular interest to us are the cases when g_v has exactly two zeroes, i.e., when it is tangent to the x axis in one of its stationary points. This holds exactly for two functions g_{v_-}, g_{v_+} highlighted in Figure 2.

Proposition 11. *There exist unique values $v_+ < v_-$ such that the function g_{v_-} takes value zero at its local maximum, and the function g_{v_+} takes value zero at its local minimum.*

The local maximum of g_{v_-} and the local minimum of g_{v_+} provide us with two *pivots*, that we call p_- and p_+ respectively. The result of this section is that the zeroes of g_v, and hence the fixpoints of h_v, always have the same position relative to the pivots p_- and p_+, for any value of v.

Proposition 12. *Let $v \in \mathbb{R}$. The function h_v has one, two, or three fixpoints. They are in $[-1, +1]$. Furthermore,*

1. *if h_v has one fixpoint x_1, then $x_1 \leq p_-$ or $p_+ \leq x_1$;*
2. *if h_v has two fixpoints $x_1 < x_3$, then $x_1 \leq p_- < p_+ \leq x_3$ or $x_1 \leq p_- < p_+ \leq x_3$;*
3. *if h_v has three fixpoints $x_1 < x_2 < x_3$, then $x_1 \leq p_- < x_2 < p_+ \leq x_3$.*

Proof sketch. The fixpoints of h_v correspond to the zeroes of g_v. Considering the intervals $I_1 = [-1, p^v_-]$, $I_2 = (p^v_-, p^v_+)$, and $I_3 = [p^v_+, +1]$, we have that Proposition 9 implies the following cases: (i) g_v has one zero x_1, and either $x_1 \in I_1$ or $x_1 \in I_3$; (ii) g_v has two zeroes x_1, x_3, and either $x_1 \in I_1$ and $x_3 \in I_2$, or $x_1 \in I_2$ and $x_3 \in I_3$; and (iii) g_v has three zeroes $x_1 \in I_1$, $x_2 \in I_2$, $x_3 \in I_3$.

In this proof sketch we discuss the case when g_v has a zero $x_1 \in I_1$. In this case, to show the proposition, it suffices to show $x_1 \leq p_-$. The idea is to relate the stationary points p^v_- and p_-. We have that g_v is an upward-right translation of g_{v_-}, since g_v has a zero $x_1 \in I_1$. Referring to Figure 2, examples of g_v are the curves above the blue curve of g_{v_-}. This in particular implies $p_- < p^v_-$. The amount of horizontal and vertical shift is $d^- = (v - v_-)/w < 0$ according to Proposition 10. The same proposition implies that d^- is the horizontal shift of the derivative, and hence $p_-^v = p_- - d^-$ by that fact that stationary points are zeroes of the derivative. Considering that $g_{v_-}(p_-) = 0$, the shift implies that $g_v(p_-^v) = -d^- > 0$. Then, according to Proposition 9, the slope $g_v'(x)$ of the curve $g_v(x)$ in the interval I_1 is bounded as $[0, 1]$, i.e., the function g_v grows sublinearly, and hence the value of g_v changes by less than $-d^-$ in the interval $[p_-, p_-^v]$ whose length is $-d^-$. Therefore the value of g_v has not reached zero yet at p_-, and hence its zero x_1 is further to the left, satisfying $x_1 \leq p_-$ as required. This concludes the proof of the considered case. The other cases can be proved using similar observations. □

4.5 Construction of the semiautomata cascade

In this section we construct a cascade $C = A_1 \ltimes \cdots \ltimes A_n$ of three-state semiautomata that homomorphically represents the target semiautomaton A_T, proving Lemma 2 and hence our central Lemma 1. The construction makes use of the preliminary results proved in the previous sections.

The construction is based on the idea that the relevant states of the prefix $P_i = N_1 \ltimes \cdots \ltimes N_i$ of the RNC+ dynamics can be grouped into 3^i classes with the homomorphism ψ treating equally all states in the same class. Recalling the equivalence relation introduced in Definition 7, our first step is to devise a function $\bar{\rho}_i$ that maps the relevant states of P_i into 3^i classes while preserving the equivalence relation, in the sense of Proposition 13. Then the homomorphism will treat the states in each class equally since it is invariant under the equivalence relation according to Proposition 7.

First we introduce an auxiliary function that categorises any real value into one of three digits, based on its position relative to the pivots p_- and p_+ introduced in Section 4.4.

Definition 8. *We define the set* $\mathbb{D} = \{1, 2, 3\}$*, and we define the function* $\kappa : \mathbb{R} \to \mathbb{D}$ *as*

$$\kappa(x) = \begin{cases} 1 & \text{if } x \leq p_-, \\ 2 & \text{if } p_- < x < p_+, \\ 3 & \text{if } p_+ \leq x. \end{cases}$$

Next we introduce a function that categorises states.

Definition 9. *Let* $i \in [n]$*. The function* $\eta_i : X'_{[i]} \to \mathbb{D}$ *is*

$$\eta_i(x_1, \ldots, x_i) = \kappa(\mathcal{S}_i^*(u_e, x_1, \ldots, x_i)).$$

Then, the function $\bar{\eta}_i : X'_{[i]} \to \mathbb{D}^i$ *is*

$$\bar{\eta}_i(x_1, \ldots, x_i) = \langle \eta_1(x_1), \ldots, \eta_i(x_1, \ldots, x_i) \rangle.$$

The function η_i takes a state $\langle x_1, \ldots, x_i \rangle$ and considers the fixpoint $\mathcal{S}_i^*(u_e, x_1, \ldots, x_i)$ to which it converges on input u_e. Then, the fixpoint is categorised by κ. We are now ready to introduce the function $\bar{\rho}_i$ mentioned above.

Definition 10. *Let* $i \in [n]$*. The function* $\bar{\rho}_i : X'_{[i]} \to \mathbb{D}$ *is*

$$\bar{\rho}_i(\mathbf{x}) = \min\{\bar{\eta}_i([\![\mathbf{x}]\!])\}.$$

Then, the function $\rho_i : X'_{[i]} \to D'_i$ *is*

$$\rho_i(\mathbf{x}) = d_i \quad \text{for} \quad \bar{\rho}_i(\mathbf{x}) = \langle d_1, \ldots, d_i \rangle.$$

The function $\bar{\rho}_i(\mathbf{x})$ returns a tuple of digits *representing* the equivalence class $[\![\mathbf{x}]\!]$. Note that $\bar{\rho}_i(\mathbf{x}) = \bar{\eta}_i(\mathbf{x})$ when $[\![\mathbf{x}]\!] = [\![\mathbf{x}]\!]_e$. We show it preserves equivalence.

Proposition 13. *For every* $i \in [n]$*, and every* $\mathbf{x}, \mathbf{y} \in X'_{[i]}$*, if* $\bar{\rho}_i(\mathbf{x}) = \bar{\rho}_i(\mathbf{y})$ *then* $\mathbf{x} \sim \mathbf{y}$.

Proof. Assuming $\bar{\rho}_i(\mathbf{x}) = \bar{\rho}_i(\mathbf{y})$, we have

$$\min\{\bar{\eta}_i([\![\mathbf{x}]\!])\} = \min\{\bar{\eta}_i([\![\mathbf{y}]\!])\} = \langle d_1, \ldots, d_i \rangle.$$

Then, there exists a pair of states $\langle x_{1,0}, \ldots, x_{i,0} \rangle \in [\![\mathbf{x}]\!]$ and $\langle y_{1,0}, \ldots, y_{i,0} \rangle \in [\![\mathbf{y}]\!]$ such that

$$\bar{\eta}_i(x_{1,0}, \ldots, x_{i,0}) = \bar{\eta}_i(y_{1,0}, \ldots, y_{i,0}) = \langle d_1, \ldots, d_i \rangle.$$

We show $\langle x_{1,0}, \ldots, x_{i,0} \rangle \sim_e \langle y_{1,0}, \ldots, y_{i,0} \rangle$, and then the proposition will follow immediately by transitivity of the equivalence relation. Let $x_{1,*}, \ldots, x_{i,*}$ and $y_{1,*}, \ldots, y_{i,*}$ be

$$\bar{\mathcal{S}}_i^*(x_{1,0}, \ldots, x_{i,0}) = \langle x_{1,*}, \ldots, x_{i,*} \rangle,$$
$$\bar{\mathcal{S}}_i^*(y_{1,0}, \ldots, y_{i,0}) = \langle y_{1,*}, \ldots, y_{i,*} \rangle.$$

It suffices to show $\langle x_{1,*}, \ldots, x_{i,*} \rangle = \langle y_{1,*}, \ldots, y_{i,*} \rangle$, which we show next by induction on i.

In the base case $i = 1$. By Proposition 5, we have that $x_{1,*}$ and $y_{1,*}$ are fixpoints of the function $h_{1,v}$ for $v = \beta_1(u_e)$. If $w_1 \in [0, 1]$, then $h_{1,v}$ has a unique fixpoint by Proposition 8, and hence $x_{1,*} = y_{1,*}$ as required. Next we consider the case when $w_1 > 1$. We have $\eta_1(x_{1,0}) = \eta_1(y_{1,0})$ and hence, by the definition of η_1, one of the three following conditions holds: (i) $x_{1,*}, y_{1,*} \leq p_-$, (ii) $p_- < x_{1,*}, y_{1,*} < p_+$, (iii) $p_+ \leq x_{1,*}, y_{1,*}$. Then, by Proposition 12, it follows that $x_{1,*} = y_{1,*}$ as required.

In the inductive case $i \geq 2$, and the inductive hypothesis is $\langle x_{1,*}, \ldots, x_{i-1,*} \rangle = \langle y_{1,*}, \ldots, y_{i-1,*} \rangle$. By Proposition 5, we have that $x_{i,*}$ is a fixpoint of the function $h_{i,v}$ for $v_x = \beta_i(u_e, x_{1,*}, \ldots, x_{i-1,*})$, and we have that $y_{i,*}$ is a fixpoint of the function $h_{i,v}$ for $v_y = \beta_i(u_e, y_{1,*}, \ldots, y_{i-1,*})$. By the inductive hypothesis, we have $v_x = v_y$, and hence let us rename them v. Thus, $x_{i,*}$ and $y_{i,*}$ are fixpoints of the same function $h_{i,v}$. If $w_i \in [0, 1]$, then $h_{i,v}$ has a unique fixpoint by Proposition 8, and hence $x_{i,*} = y_{i,*}$ as required. Next we consider the case when $w_i > 1$. We have $\eta_i(x_{1,0}, \ldots, x_{i,0}) = \eta_i(y_{1,0}, \ldots, y_{i,0})$, and hence, by the definition of η_i, we have that one of the three following conditions holds: (i) $x_{i,*}, y_{i,*} \leq p_-$, (ii) $p_- < x_{i,*}, y_{i,*} < p_+$, (iii) $p_+ \leq x_{i,*}, y_{i,*}$. Since $x_{i,*}$ and $y_{i,*}$ are fixpoints of $h_{i,v}$ as argued above, by Proposition 12, it follows that $x_{i,*} = y_{i,*}$ as required. This concludes the proof. □

Construction. We are now ready to define the cascade C. For $i \in [n]$, the semiautomaton A_i of C is $A_i = \langle \Sigma_i, Q_i, \delta_i \rangle$ where the alphabet and states are

$$\Sigma_i = \Sigma \times Q_1 \times \cdots \times Q_{i-1}, \qquad Q_i = \rho_i\big(X'_{[i]}\big),$$

and the transition function is defined as

$$\delta_i(d_i, \langle \sigma, d_1, \ldots, d_{i-1} \rangle) = \rho_i\big(\bar{f}_i(\mathbf{x}, u_\sigma)\big), \tag{1}$$

for any u_σ such that $\lambda(u_\sigma) = \sigma$ and any $\mathbf{x} \in X'_{[i]}$ such that $\rho(\mathbf{x}) = \langle d_1, \ldots, d_i \rangle$ if there is such an $\mathbf{x}$, and otherwise

$$\delta_i(d_i, \langle \sigma, d_1, \ldots, d_{i-1} \rangle) = d_i. \tag{2}$$

The transition function is indeed a function, in light of the following proposition.

Proposition 14. *For every $u, v \in U$, and every $\mathbf{x}, \mathbf{y} \in X'_{[i]}$, if $\lambda(u) = \lambda(v)$ and $\bar{\rho}_i(\mathbf{x}) = \bar{\rho}_i(\mathbf{y})$ then*

$$\rho_i\big(\bar{f}_i(\mathbf{x}, u)\big) = \rho_i\big(\bar{f}_i(\mathbf{y}, v)\big).$$

Proof. By Proposition 13, we have $\mathbf{x} \sim \mathbf{y}$. Then by the definition of $\sim$, we have $\bar{f}_i(\mathbf{x}, u) \sim \bar{f}_i(\mathbf{y}, v)$. Then the proposition follows immediately by the definition of ρ_i. □

The states of A_i are the digits returned by ρ_i on the relevant states of P_i. The transition function is defined by two cases. Equation (1) defines it in terms of the dynamics function $\bar{f}_i$ of P_i, by applying it to an RNC+ state and input that are mapped to the current state and input of A_i. To have a totally-defined transition function, Equation (2) completes the definition with a dummy choice of the successor state, which is argued below to be irrelevant. The specific choice of u_σ and $\mathbf{x}$ in Equation (1) among the possible ones does not affect the outcome of the transition function, by the invariance property described in Proposition 14.

The resulting cascade is $C = \langle \Sigma, Q_C, \delta_C \rangle$ with states $Q_C = Q_1 \times \cdots \times Q_n$ and transition function

$$\delta_C(\langle d_1, \dots, d_n \rangle, \sigma) = \langle \delta_1(d_1, \sigma_1), \dots, \delta_n(d_n, \sigma_n) \rangle,$$
$$\text{with } \sigma_i = \langle \sigma, d_1, \dots, d_{i-1} \rangle.$$

Finally we are ready to show that the constructed cascade C captures the target semiautomaton A_T, so proving the main lemma.

Lemma 2. *It holds that C homomorphically represents A_T.*

Proof. Let $Q'_C = \bar{\rho}_n(X')$. We have that $\delta_C(Q'_C, \Sigma) \subseteq Q'_C$, since $f(X', U) \subseteq X'$ because X' are states of the subdynamics D'. Thus $C' = \langle \Sigma, Q'_C, \delta_C \rangle$ is a subsemiautomaton of C. Note that, on states Q'_C, the transition function δ_C is defined by Eq. (1).

It suffices to show a homomorphism ψ' from C' to A_T. We define $\psi'(\mathbf{d}) = \psi(\mathbf{x})$ for any choice of $\mathbf{x} \in X'$ such that $\bar{\rho}_n(\mathbf{x}) = \mathbf{d}$. Note that ψ' is indeed a function, since, by Proposition 13, ψ is invariant under the equivalence $\sim$, and every $\mathbf{x} \in X'$ satisfying $\bar{\rho}_n(\mathbf{x}) = \mathbf{d}$ is from the same equivalence class $[\![\mathbf{x}]\!]$. We show that $\psi' : Q'_C \to Q_T$ is a homomorphism from C' to A_T. First, ψ' is continuous as required, since ψ is continuous. Second, we argue that ψ' is surjective as required. Let $q \in Q_T$. It suffices to show some $\mathbf{d} \in Q'_C$ such that $\psi'(\mathbf{d}) = q$. We have that ψ is surjective, and hence there exists $\mathbf{x} \in X'$ such that $\psi(\mathbf{x}) = q$. We have $\bar{\rho}_n(\mathbf{x}) = \mathbf{d} \in Q'_C$, and hence $\psi'(\mathbf{d}) = \psi(\mathbf{x}) = q$ by the definition of ψ'. Third, we argue that ψ' satisfies the homomorphism condition. Let $\mathbf{d} \in Q'_C$, let $\mathbf{x}$ be the such that $\rho(\mathbf{x}) = \mathbf{d}$, let $\sigma \in \Sigma$, and let u_σ be such that $\lambda_\Sigma(u_\sigma) = \sigma$. Then,

$$\begin{aligned}
\psi'(\delta'_C(\mathbf{d}, \sigma)) &= \psi'(\bar{\rho}_n(f(\mathbf{x}, u_\sigma))) \\
&= \psi(f(\mathbf{x}, u_\sigma)) \\
&= \delta_\Sigma(\psi(\mathbf{x}), u_\sigma) \\
&= \delta_T(\psi(\mathbf{x}), \sigma) \\
&= \delta_T(\psi'(\mathbf{d}), \sigma).
\end{aligned}$$

Therefore C homomorphically represents A_T. □

5 Related Work

The ability of non-differentiable RNNs to capture formal languages is discussed in (Kleene 1956; Nerode and Sauer 1957; Minsky 1967). These are networks such as the ones from (McCulloch and Pitts 1943), and their expressivity coincides with the regular languages. In this paper and the rest of this section we focus on differentiable neural networks.

The Turing-completeness capabilities of RNNs as an *offline model of computation* are studied in (Siegelmann and Sontag 1995; Kilian and Siegelmann 1996; Hobbs and Siegelmann 2015; Chung and Siegelmann 2021). In this setting, an RNN is allowed to first read the entire input sequence, and then return the output after an arbitrary number of iterations, triggered by blank inputs. This differs from our setting, which focuses on the capabilities of RNNs as an *online model of computation*, where the input sequence is processed one element at a time, outputting a value at every step. This is the way they are used in many practical applications such as Reinforcement Learning, cf. (Bakker 2001; Hausknecht and Stone 2015; Ha and Schmidhuber 2018; Kapturowski et al. 2019).

A form of asymptotic expressivity for RNNs is studied in (Merrill et al. 2020). They consider the expressivity of RNNs when their weights tend to infinity, which effectively makes them finite-state for squashing activation functions such as tanh, yielding an expressivity within the regular languages. In our work we consider the expressivity of networks with their actual finite weights.

Transformers are another class of neural networks for sequential data (Vaswani et al. 2017). They are a *non-uniform model of computation*, in the sense that inputs of different lengths are processed by different networks. This differs from RNNs which are a *uniform model of computation*. The expressivity of transformers has been studied by relating them to families of Boolean circuits and logics on sequences (Hahn 2020; Hao, Angluin, and Frank 2022; Merrill, Sabharwal, and Smith 2022; Liu et al. 2023; Chiang, Cholak, and Pillay 2023; Merrill and Sabharwal 2023).

6 Conclusions and Future Work

We have extended the understanding of the expressivity landscape for RNC. Specifically, we have shown that the class of formal languages with an identity element that can be recognised by RNC+ is the star-free regular languages. This reinforces the fact that RNC+ is a strong candidate for learning temporal patterns captured by many well-known temporal formalisms.

There are several interesting directions for future work. The main open question regards the expressivity of RNC+ beyond regular and beyond identity, the white area in Figure 1. No results are known for these languages. A second open question regards the expressivity of RNC when recurrent weights can be negative. According to existing results, negative weights extend the expressivity of RNCs beyond star-free. However, no precise characterisation is known. Third, it is interesting future work to study the expressivity of RNC with other activation functions such as logistic curve, ReLU, and GeLU.

Acknowledgments

Alessandro Ronca is supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 852769, ARiAT). For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission.

References

Bakker, B. 2001. Reinforcement learning with long short-term memory. In *NeurIPS*.

Chiang, D.; Cholak, P.; and Pillay, A. 2023. Tighter bounds on the expressivity of transformer encoders. In *ICML*.

Chung, S., and Siegelmann, H. T. 2021. Turing completeness of bounded-precision recurrent neural networks. In *NeurIPS*.

De Giacomo, G., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*.

Fahlman, S. E. 1990. The recurrent cascade-correlation architecture. In *NIPS*.

Giles, C.; Chen, D.; Sun, G.-Z.; Chen, H.-H.; Lee, Y.-C.; and Goudreau, M. 1995. Constructive learning of recurrent neural networks: Limitations of recurrent cascade correlation and a simple solution. *IEEE Trans. Neural Netw.* 6(4).

Ginzburg, A. 1968. *Algebraic Theory of Automata*. AP.

Ha, D., and Schmidhuber, J. 2018. Recurrent world models facilitate policy evolution. In *NeurIPS*.

Hahn, M. 2020. Theoretical limitations of self-attention in neural sequence models. *Trans. Assoc. Comput. Linguist.* 8.

Hao, Y.; Angluin, D.; and Frank, R. 2022. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Trans. Assoc. Comput. Linguist.* 10.

Hausknecht, M. J., and Stone, P. 2015. Deep recurrent Q-learning for partially observable MDPs. In *AAAI Fall Symp.*

Hobbs, J. N., and Siegelmann, H. T. 2015. Implementation of universal computation via small recurrent finite precision neural networks. In *IJCNN*.

Kapturowski, S.; Ostrovski, G.; Quan, J.; Munos, R.; and Dabney, W. 2019. Recurrent experience replay in distributed reinforcement learning. In *ICLR*.

Kilian, J., and Siegelmann, H. T. 1996. The dynamic universality of sigmoidal neural networks. *Inf. Comput.* 128(1).

Kleene, S. C. 1956. Representation of events in nerve nets and finite automata. *Automata studies* 34.

Knorozova, N. A., and Ronca, A. 2024a. On the expressivity of recurrent neural cascades. In *AAAI*.

Knorozova, N. A., and Ronca, A. 2024b. On the expressivity of recurrent neural cascades with identity. *arXiv* 2405.11657.

Kremer, S. C. 1996. Comments on "Constructive learning of recurrent neural networks: Limitations of recurrent cascade correlation and a simple solution". *IEEE Trans. Neural Netw.* 7(4).

Liu, B.; Ash, J. T.; Goel, S.; Krishnamurthy, A.; and Zhang, C. 2023. Transformers learn shortcuts to automata. In *ICLR*.

Manna, Z., and Pnueli, A. 1991. Completing the temporal picture. *Theor. Comp. Sci.* 83(1).

McCulloch, W. S., and Pitts, W. 1943. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* 5(4).

McNaughton, R., and Papert, S. A. 1971. *Counter-Free Automata*. The MIT Press.

Merrill, W., and Sabharwal, A. 2023. A logic for expressing log-precision transformers. In *NeurIPS 2023*.

Merrill, W.; Weiss, G.; Goldberg, Y.; Schwartz, R.; Smith, N. A.; and Yahav, E. 2020. A formal hierarchy of RNN architectures. In *ACL*.

Merrill, W.; Sabharwal, A.; and Smith, N. A. 2022. Saturated transformers are constant-depth threshold circuits. *Trans. Assoc. Comput. Linguist.* 10.

Minsky, M. L. 1967. *Computation: Finite and Infinite Machines*. Prentice-Hall.

Nerode, A., and Sauer, B. P. 1957. Fundamental concepts in the theory of systems. Technical Report 57-624, Wright Air Development Center.

Reed, R., and Marks II, R. J. 1999. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. MIT Press.

Schützenberger, M. P. 1965. On finite monoids having only trivial subgroups. *Inf. Control.* 8(2).

Shin, H.-C.; Roberts, K.; Lu, L.; Demner-Fushman, D.; Yao, J.; and Summers, R. M. 2016. Learning to read chest X-rays: Recurrent neural cascade model for automated image annotation. In *CVPR*.

Siegelmann, H. T., and Sontag, E. D. 1995. On the computational power of neural nets. *J. Comput. Syst. Sci.* 50(1).

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is all you need. In *NeurIPS*.

Wang, J.; Zheng, V. W.; Liu, Z.; and Chang, K. C.-C. 2017. Topological recurrent neural network for diffusion prediction. In *ICDM*.

Werbos, P. 1990. Backpropagation through time: What it does and how to do it. *Proc. IEEE* 78(10).

Xu, Z.; Sun, C.; Ji, T.; Manton, J. H.; and Shieh, W. 2020. Cascade recurrent neural network-assisted nonlinear equalization for a 100 Gb/s PAM4 short-reach direct detection system. *Optics Letters* 45(15).

Zhang, D.; Yao, L.; Zhang, X.; Wang, S.; Chen, W.; Boots, R.; and Benatallah, B. 2018. Cascade and parallel convolutional recurrent neural networks on EEG-based intention recognition for brain computer interface. In *AAAI*.

Zhu, L.; Huang, L.; Fan, L.; Huang, J.; Huang, F.; Chen, J.; Zhang, Z.; and Wang, Y. 2020. Landslide susceptibility prediction modeling based on remote sensing and a novel deep learning algorithm of a cascade-parallel recurrent neural network. *Sensors* 20(6).