Copyright Statement

Plymouth University

Doctoral Thesis

Dynamical Systems Theory for Transparent Symbolic Computation in Neuronal Networks

Author:
Giovanni Sirio Carmantini

Supervisor:
Dr. Serafim Rodrigues

A thesis submitted to Plymouth University
in partial fulfilment for the degree of

Doctor of Philosophy

in the

Centre for Robotics and Neural Systems
School of Computing, Electronics and Mathematics

March 2017

Acknowledgements

One thousand one hundred and three days have passed since I started my PhD journey and, as I write these acknowledgements, I can only find good memories of my time here. That is a pretty good stretch of continued happiness to achieve, and I owe every single minute of it to the fantastic people that happened to cross my path. I am grateful for that.

First of all, I am grateful to my supervisor Dr. Serafim Rodrigues. When I applied for a PhD position, I chose an advisor rather than a project. Serafim supported me at every step of the journey, with a great sense of responsibility and humanity. His intelligence, passion, and perspective on life inspire me. I cannot thank him enough for all the guidance he has provided me over the last three years, and for setting up the best network a PhD student could wish for: I am grateful to Dr. Fabio Schittler Neves, who always met me with a smile and sharp advice; Dr. Mathieu Desroches, whose sense of humor is only second to his insight and kindness; Dr. Peter beim Graben, for his acuity, clarity of thought and warmth; Dr. Martin Krupa for his help and support.

My sincere thanks also goes to my second supervisor Prof. Angelo Cangelosi, who was always there to advise and support me, with friendly words, tea and biscuits.

I thank my labmates Valerio Biscione, Ricardo De Azambuja, Massimiliano Patacchiola and Debora Zanatto for their sincere friendship, their intelligence, jokes, and support, all of which made the office feel like home. In particular, I thank Debora for her positive personality, for her empathy and ability to listen; I thank Ricardo for being as wise as he is light-hearted, and for the many great pieces of advice he shared with me during this time; to Massimiliano I owe many laughs, and many intelligent conversations. I thank both Ricardo and Massimiliano for bringing to the office the many unattended sweets that have propelled me during this last month. I also thank Valerio for his absence during the writing of this thesis: if he hadn't left for Bristol, I would have rather joked and reasoned with him than work. His friendship is one of the best things Plymouth will leave me.

I would also like to thank my family, who made me who I am and who I owe everything in my life. Thank you for being so wonderful, I feel fortunate to have you.

Finally, I thank my girlfriend Giulia, who every day teaches me how to enjoy life's every little moment.

Declaration of Authorship

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other University award without prior agreement of the Graduate Sub-Committee.

Work submitted for this research degree at the Plymouth University has not formed part of any other degree either at Plymouth University or at another establishment.

This study was financed with the aid of a studentship from the University of Plymouth.

Relevant scientific seminars and conferences were regularly attended at which work was often presented; external institutions were visited for consultation purposes and papers prepared for publication.

Education:

- MATH3405 Nonlinear Systems,
  Plymouth University (Plymouth, United Kingdom, 2015)
- Learning and Teaching for General Teaching Associates,
  Plymouth University (Plymouth, United Kingdom, 2015),
- Postgraduate Certificate in Academic Practice, General Teaching Associates course,
  Plymouth University (Plymouth, United Kingdom, 2016),
- LxMLS Lisbon Machine Learning School,
  Instituto Superior Tecnico (Lisbon, Portugal, 2016).

Publications:

- Carmantini, G.S., beim Graben, P., Desroches, M., and Rodrigues, S. (in print).
  A modular architecture for transparent computation in recurrent neural networks.
  Neural Networks.

– Carmantini, G.S., beim Graben, P., Desroches, M., and Rodrigues, S. (2015).
Turing computation with recurrent artificial neural networks.
In *Proceedings of the NIPS Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches*, pages 5 – 13.
arXiv:1511.01427 [cs.NE].

Conferences Attended:

– The annual meeting of the Cognitive Science society,
CogSci (Quebec City, Canada, 2014)

– Workshop on Heteroclinic Dynamics in Neuroscience,
HDN (Nice, France, 2015)

– Neural Information Processing Systems,
NIPS (Montreal, Canada, 2015)

Word count of main body of thesis: 35948 words

Signed: _____

Date: _____

"What I cannot create I do not understand."

Richard Feynman

Abstract

Giovanni Sirio Carmantini

Dynamical Systems Theory for Transparent Symbolic Computation in Neuronal Networks

In this thesis, we explore the interface between symbolic and dynamical system computation, with particular regard to dynamical system models of neuronal networks. In doing so, we adhere to a definition of computation as the physical realization of a formal system, where we say that a dynamical system performs a computation if a correspondence can be found between its dynamics on a vectorial space and the formal system's dynamics on a symbolic space. Guided by this definition, we characterize computation in a range of neuronal network models. We first present a constructive mapping between a range of formal systems and Recurrent Neural Networks (RNNs), through the introduction of a *Versatile Shift* and a modular network architecture supporting its real-time simulation. We then move on to more detailed models of neural dynamics, characterizing the computation performed by networks of delay-pulse-coupled oscillators supporting the emergence of heteroclinic dynamics. We show that a correspondence can be found between these networks and Finite-State Transducers, and use the derived abstraction to investigate how noise affects computation in this class of systems, unveiling a surprising facilitatory effect on information transmission. Finally, we present a new dynamical framework for computation in neuronal networks based on the *slow-fast* dynamics paradigm, and discuss the consequences of our results for future work, specifically for what concerns the fields of interactive computation and Artificial Intelligence.

Contents

List of Figures

List of Tables

Abbreviations

ANN	Artificial Neural Network
BSL	Branch Selection Layer
CFG	Context-Free Grammar
CL	Configuration Layer
CPG	Central Pattern Generator
DCFG	Deterministic Context-Free Grammar
DFSM	Deterministic Finite-State Machine
DPDA	Deterministic Push-Down Automaton
EEG	ElectroEncephaloGraphy
ERP	Event-Related brain Potentials
FSM	Finite-State Machine
FST	Finite-State Transducer
GS	Generalized Shift
LFP	Local Field Potentials
LIF	Leaky Integrate-and-Fire (oscillators)
LTL	Linear Transformation Layer
MCL	Machine Configuration Layer
MI	Mutual Information
MIR	Mutual Information Rate
NCFG	Nondeterministic Context-Free Grammar
NDA	Nonlinear Dynamical Automaton
NFSM	Nondeterministic Finite-State Machine
NPDA	Nondeterministic Push-Down Automaton
PCA	Principal Component Analysis
PDA	Push-Down Automaton
RNN	Recurrent Neural Network
SNR	Signal-to-Noise Ratio
SSM	State-Switching Machine
synth-ERP	Synthetic Event-Related brain Potential
TDR	Top-Down Recognizer
TM	Turing Machine
UTM	Universal Turing Machine
VS	Versatile Shift

Dedicated to my parents

Introduction

> We can only see a short distance ahead,
> but we can see plenty there that needs to be done.
>
> Alan Turing (1950)

The concept of computation is at the center of many of the most important theoretical and technological advancements of the last century. Digital computers surround us and support us in almost every activity and, as the field of Machine Learning advances at an unprecedented speed, we witness the dawning of the age of intelligent machines.

The scientific foundations for the digital revolution were laid almost a century ago, as a result of Alan Turing's groundbreaking work. In order to solve a long-standing problem in mathematical logic, the brilliant British scientist imagined a hypothetical machine that could read and write symbols on a tape, modelling the action of a person performing calculations with pen and paper (Turing, 1937). The machine proved to be an exceptionally powerful conceptual tool, marking the origins of the *Theory of Computation*, and inspiring the creation of the first re-programmable digital computers. In particular, through the Turing Machine, computation came to be defined as the mechanical application of a finite sequence of symbolic operations to an input string of symbols, in order to produce a symbolic output. The success of digital computers and of the theory that describes their operation has been so striking that computation today is almost exclusively understood as Turing computation.

In recent years, however, technological progress and theoretical developments in disciplines such as Biology, Neuroscience, and Physics have started to bring attention to the limits

of the traditional understanding of computation as a sequential input-output process of symbolic manipulation.

As a first problematic point, real-world digital computing systems have evolved far beyond what the classical Theory of Computation can describe. In an age where computers interact with each other and their environment constantly and asynchronously, it is not clear how their action can be modelled as discrete input-output relations, especially when the computation they perform does not "halt". A digital thermostat, for example, constantly reads and reacts to its environment in a non-terminating interaction; moreover, by reacting to its environment, the thermostat also modifies it, thus affecting its future readings. These forms of open-ended and interactive computation, while ubiquitous in our modern world, are not captured by the classical framework of Turing Computation.

Secondly, as our understanding of many important biological and physical systems increases, we encounter phenomena which call for a description in terms of information processing, but which do not quite fit in the categories of the sequential rule-based manipulation of strings of symbols (i.e. Turing Computation). That is, we may talk for example of a neuronal assembly "computing" the movement direction of a visual stimulus (Emerson et al., 1992), or "information processing" of auditory stimuli (Näätänen, 1990), or "storing" data in the connection patterns of its neurons (Chklovskii et al., 2004). Yet, it is not clear how these phenomena may relate to our understanding of computation, as rooted in the classical Theory of Computation. In such systems, computation is realized within the dynamical evolution of a physical process, best modeled through the framework of difference/differential equations, the realm of Dynamical Systems Theory, rather than rewriting rules acting on symbols. When these systems are analyzed for the forms of computation they can support (rather than for their biological or physical significance), their study falls under the umbrella term of *natural computation*.

Recent years have seen the characterization of a number of dynamical mechanism for natural computation. Of particular interest for this work are systems modelling neuronal dynamics, for which many such mechanisms have been uncovered for the transmission (Borisyuk and Borisyuk, 1997, Mazor and Laurent, 2005, Neves and Timme, 2009, 2012,

Wordsworth and Ashwin, 2008), transformation (Ashwin and Borresen, 2005, Jaeger and Haas, 2004, Larger et al., 2012, LukošEvičIus and Jaeger, 2009, Maass et al., 2002, Rabinovich et al., 2008a) and storage of information (Bick and Rabinovich, 2009, Hopfield, 1982). In parallel with the proliferation of dynamical paradigms for computation, important ongoing efforts are in place for the creation of a general theory of computation in dynamical systems (Bournez and Campagnolo, 2008, Dambre et al., 2012, Langton, 1990, Orponen, 1997, Siegelmann and Fishman, 1998, Stepney, 2012).

Unfortunately, the present state of our understanding of computation is a set of insular and contextualized descriptions. As the limits of the classic approaches to computation become more evident, we witness a disconnect between how we think of information processing in symbolic/digital systems, and how we think of it in dynamical/analog systems; between the theory of idealized computing machines, and the far more complex reality of how they operate in the world.

Nevertheless, there have been encouraging achievements towards bridging the gaps in our knowledge and pursue a more general theory of computation. Crucial work has been done, for example, in the field of interactive computation (Cabessa and Siegelmann, 2012, Goldin et al., 2006, Wegner, 1998) and in the integration between dynamical system and symbolic computation (beim Graben, 2008, beim Graben et al., 2008, 2004, beim Graben and Potthast, 2014, Branicky, 1995, Cabessa and Villa, 2012, 2013, MacLennan, 2004, Moore, 1990, 1991, Siegelmann and Sontag, 1995, Tabor, 2009, Tabor et al., 1997).

In this work, we take heed of these crucial integrative achievements, and explore the common area where symbol manipulation and the dynamics of neuronal systems meet together, with the hope to contribute to a richer and more unified understanding of computation. We do so by adopting the general framework proposed by MacLennan (2004), who proposes an extended definition of computation as a "physical process the purpose of which is the abstract manipulation of abstract objects", where a system is seen as performing a computation if it is a physical (possibly approximate) realization of an abstract mathematical structure – a formal system – and if it serves a function which depends only on its

formal properties, rather than its physical instantiation. [1] In this framework, if a system is computational, the specific physical surrogates by which its formal properties are instantiated do not matter. In fact, given that the formal properties of the system stay the same, then a computational system will serve its purpose (whatever this might be) just as well. As an example, if we say that a neuronal assembly in the brain computes the orientation of a stimulus in a region of the visual field, then in terms of computation it should not matter to the rest of the brain whether we substitute the neuronal assembly with a silicon microprocessor implementing the same functionality: assuming that the physical surrogates are appropriately transduced (such that the interaction of the microprocessor with the rest of the brain is the same as the one of the neural assembly it substitutes), it will serve the same purpose – computing orientation – just as well. As a second example, if the purpose of a system is to perform the addition of two natural numbers, it does not matter from a computation standpoint whether the system is a simple abacus, a complex mechanical device such as the Pascaline, or a digital one such as a modern calculator, as far as a (possibly approximate) correspondence between physical quantities in such diverse systems and the addition of natural numbers can be found.

In adopting MacLennan's broader definition of computation (admittedly imprecise, but nevertheless useful here as a conceptual framework), in this work we characterize computation in a variety of dynamical systems by showing how they instantiate the abstract properties of some formal system[2], thus defining a correspondence between the two (see Figure 1). That is, to show that some dynamical system performs a given form of computation (e.g. a Finite-State computation), we can present a correspondence between the structure and dynamics of the dynamical system and the formal system that defines that form of computation (e.g. a Finite-State Machine). Similarly, to demonstrate how a form

[1] The word "purpose", while problematic, is there to avert the pitfalls of pan-computationalism, or the idea for which every physical process can be seen as a form of computation.

[2] A formal system here is intended as a mathematical object defining a set of symbols, a grammar specifying how well-formed formulas in the system are constructed, a set of axioms which are themselves well-formed formulas, and a set of inference rules combining well-formed formulas to obtain new well-formed formulas. Every model of symbolic computation can also be expressed as a formal system, but a formal system need not necessarily be a model of computation. What distinguishes a model of computation from the more general class of formal systems is that a model of computation is explicitly constructed and used in the study of computation, which is not true in general for formal systems. Yet, it is always possible to study the class of functions that can be computed by any given formal system, such that the distinction between formal models and models of computation is really only of the use which is made of the model in practice.

of computation can be performed by a dynamical system, can we show how to instantiate the structure and evolution of the formal system which defines that form computation in the structure and dynamics of the dynamical system. We will in fact introduce results in both directions, where we i) construct dynamical systems that realize a range of formal systems, and ii) characterize the formal properties of a class of dynamical systems to specify the forms of computation they support. We do so in order to tackle the following research question, which we put at the center of this work: Is it possible to define a constructive mapping between formal models of symbolic computation and models of neural dynamics such that the defined mapping is general (i.e. it can be applied to a wide range of symbolic models of computation), and the models of neural dynamics are biologically plausible?

Structure of this thesis

The thesis is divided in two parts. The first part sets the theoretical background of our work with three Chapters, each containing the relevant literature review:

- In Chapter 1, we present some fundamental systems from an important branch of the classical Theory of Computation, i.e. the theory of automata and formal languages. We will use these objects to support the modelling of different forms of symbolic computation in the rest of the thesis.
- In Chapter 2, we provide key definitions and concepts from Dynamical Systems Theory, such as continuous- and discrete-time dynamical systems, fixed points, and heteroclinic dynamics.
- In Chapter 3, we discuss the theory of Symbolic Dynamics, i.e. the theoretical apparatus that allows us to bridge the realm of symbolic computation to that of dynamical systems.

The second part will then dedicate efforts to the presentation and discussion of the original theoretical advancements we put forward, with each Chapter containing the relevant literature review:

– In Chapter 4, we introduce a formal system allowing for the parsimonious real-time simulation of a range of models of computation, i.e. the Versatile Shift. We then show that the Gödelization of Versatile Shifts defines piecewise affine-linear dynamical systems on the unit square, i.e. Nonlinear Dynamical Automata. By presenting a constructive mapping between Nonlinear Dynamical Automata and Recurrent Neural Networks (RNNs), we are able to relate symbolic dynamics in the Versatile Shift formal system to vectorial dynamics in a neural system. The network architecture defined by the mapping is characterized by its granular modularity; coupled with a transparent implementation of the original symbolic dynamics, our work allows for the construction of neural implementations of interactive models of computation. We demonstrate this possibility with two examples. In one example we construct a RNN Central Pattern Generator from a Finite-State Machine, and show that it supports continuous interaction with the environment through a non-terminating computation (the generation of the pattern). In a second example we demonstrate the mapping of an Interactive Automata Network to RNN dynamics, implementing the interactive communication between different automata as interactive communication between sub-networks in the RNN. The constructive mapping opens the possibility of preliminary correlational studies between real neural dynamics and RNN implementations of Versatile Shifts, allowing the characterization of computation in the real system (as per MacLennan's framework). Elements from this Chapter have been published in Carmantini et al. (2016) and Carmantini et al. (2015).

  As we argue at the end of this Chapter, there are important concerns about the biological plausibility of RNNs and of our architecture in particular, especially in relation to the disruptive effect of noise on the dynamics of these networks. In order to find more suitable candidates for the definition of a constructive mapping between formal systems and neural models, we turn in the following Chapter to a more robust and biologically plausible class of neural models which are known to support Finite-State computation, a simple form of symbolic computation. Specifically, we test the suitability of such models to robustly support symbolic computation by exploring the effect of noise on their dynamics.

– In Chapter 5 we first characterize the computation performed by systems of oscillators supporting the emergence of heteroclinic networks, by defining a correspondence between their dynamics and the symbolic dynamics of Finite-State Transducers, setting the language needed for the subsequent analyses. We then study the information transmission capabilities of this class of systems when a noise source is present, showing that the state-switching they implement becomes probabilistic, as opposed to the deterministic switching in the absence of noise. We also show that for a certain range of noise levels, the probabilistic state-switching allows for a more efficient transmission of information by the system. This is particularly relevant when considering that heteroclinic dynamics is thought to have a key role in the sensory processing of some animals. Given that noise adds non-determinism to the dynamics of these models, in future work we plan to define a correspondence between noisy heteroclinic systems and probabilistic formal systems, such as probabilistic Finite-State Transducers or Markov Chains. Elements from this Chapter are part of a paper which is being prepared for submission.

  While this class of models of neural dynamics is of higher biological relevance than that studied in Chapter 4, there are still key concerns about the strict set of conditions in which the heteroclinic dynamics they support can be maintained. Specifically, heteroclinic dynamics is only possible when perturbations to the system (whether from noise or input sources) are extremely small. Additionally, the state switching supported by this class of systems is asymptotic in absence of noise, i.e. a state is only reached in the limit as time goes to infinity. These characteristics cast doubts about the biological plausibility of this type of dynamics. For this reason, in Chapter 6 we further our search of viable candidates for the definition of a general mapping between formal systems and neural models, by exploring the dynamics of a class of systems with even stronger biological plausibility.

– In Chapter 6, we present preliminary numerical evidence for a new framework for computation in dynamical systems to be explored in future work, which implements the dynamic switching of states while not suffering from the drawbacks of systems in which the state switching is based on heteroclinic dynamics. In particular, the new

framework relies on smooth excitable neural models, which are the most appropriate level of abstraction with regards to the modelling of electrophysiological measures from real neural systems. We believe that a correspondence can be found between the dynamics of some *slow-fast* neural systems (an important class of smooth excitable neural models) and Finite-State computation.

– Finally, in Chapter 7 we summarize the contributions presented in this thesis, and discuss the implications of our work for the field of Artificial Intelligence, with special regards to the formalization of interaction in intelligent systems.

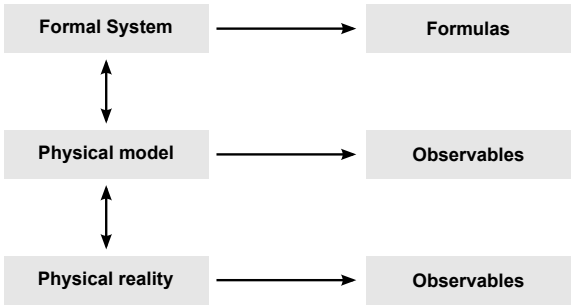


FIGURE 1: **Characterization of computation as physical realization of a formal system.** To characterize computation in physical and biological systems, we create mathematical models that capture their dynamics, and define a correspondence between these and formal systems. In this way, we are able to interpret observables from measurements of the physical world in terms of the underlying computation they reflect.

Chapter 1

Symbolic Computation

As discussed in the Introduction, computation is normally understood in terms of algorithms, rewriting systems and automata. In the rest of this work we will make use of automata models to characterize computation in a range of neuronal network models. For this reason, this Chapter presents an introduction to some important classes of automata and formal languages. Subsequently, we will briefly discuss the relation between symbolic computation and information processing, and introduce important measures from the classical Information Theory (Shannon, 1948) which we will use in later Chapters to quantify the transmission of information in a class of neural systems realizing symbolic computation.

1.1 Automata and Formal Languages Theory

We will now discuss some important models of computation from the classical Theory of Computation. For a more in-depth introductory treatment of these models, we point the reader to the excellent books by Sipser (2006) and Hopcroft and Ullman (1979).

1.1.1 Finite-State Machines

The dynamics of many real-world systems can be effectively abstracted in terms of transitions between a finite set of discrete states, such that the state-to-state transition at each

time step only depends on some current *input* to the system, where the set of possible inputs is finite; this abstraction defines a class of models known as *Finite-State Machines* (FSMs; McCulloch and Pitts, 1943). FSMs are very much relevant for modeling in a broad range of fields, including but not limited to computer science, biology, physics and engineering.

Definition 1.1. A Deterministic Finite-State Machine (DFSM, or simply FSM) can be formally defined as a 5-tuple $M_{\text{FSM}} = (Q, \mathbf{T}, q_0, F, \delta)$, where

1. Q is a finite set of machine states, or *control* states
2. $\mathbf{T}$ is a finite set of symbols known as the input alphabet,
3. $q_0 \in Q$ is a distinguished state from the set of states known as *starting* state, i.e. the state which the machine is in at the beginning of its computation,
4. $F \subseteq Q$ is a set of final (or *accept*) states,
5. $\delta : Q \times \mathbf{T} \to Q$ is a transition function. Specifically, δ can be defined as follows:

$$\delta : (q_t, d_{0_t}) \mapsto q_{t+1}, \tag{1.1}$$

where $q_t, q_{t+1} \in Q$ are states, and $d_{0_t} \in \mathbf{T}$ is an input symbol.

A FSM acts on input strings of symbols. Given an input string $s = d_0 \dots d_n$ a FSM sequentially consumes (reads and discards) an input symbol d_t from the string at each computation step t and, given the current read symbol and its current state q_t, the machine transitions to a new state $q_{t+1} = \delta(q_t, d_t)$ as prescribed by its transition function. f the last state reached by the machine once all symbols in the input string are consumed is in F, i.e. the set of final states, the machine is said to *accept* the input string.

Let us now give an example of a FSM. For this example, we present a FSM accepting all binary strings ending in 1. This machine can be defined as $M_{end1} = (Q, \mathbf{T}, q_0, F, \delta)$ where $Q = \{q_0, q_1\}$ is the set of states, $\mathbf{T} = \{\mathtt{0}, \mathtt{1}\}$ is the binary input alphabet, $F = \{q_1\}$ is the

set of accept states (with q_1 as the only accept state) and where δ is defined such that

$$\delta : \begin{cases} (q_0, \mathtt{0}) \mapsto q_0, \\ (q_0, \mathtt{1}) \mapsto q_1, \\ (q_1, \mathtt{0}) \mapsto q_0, \\ (q_1, \mathtt{1}) \mapsto q_1. \end{cases}$$

Let us present this machine an input string $s = \mathtt{101}$. The machine starts in state q_0, i.e. the starting state. It then reads the first symbol in s, i.e. a $\mathtt{1}$. Given the current read symbol $\mathtt{1}$ and the current state q_0, the machine transitions to a new state q_1, as instructed by its transition function δ. In fact, $\delta(q_0, \mathtt{1}) = q_1$. The machine is now ready to consume the second symbol in s, which is a $\mathtt{0}$. By application of the δ function, the machine now transitions to state q_0. Finally, by consuming the last symbol in s, a $\mathtt{1}$, the machine transitions again to state q_1. As the input string has now been completely consumed, q_1 is the last state reached by the machine. Given that $q_1 \in F$, the set of accept states, the machine is said to accept the input string s.

A second important flavor of FSMs is that of Nondeterministic FSMs (NFSMs). In the Nondeterministic case, the transition function δ is defined as a function $\delta : Q \times \mathbf{T} \to P(Q)$ that maps each state/symbol pair (q_t, d_{0_t}) to a set of states $O \in P(S)$, rather than a single state as in the deterministic case. $P(S)$ is here the powerset of a set S, defined as $P(S) = \{O : O \subset S\}$), i.e. the set of all possible subsets of S. In this way, a NFSM is not restricted in performing a single transition at each computation step, but can split in a set of parallel processes, each performing one of a set of possible transitions. Each of these processes can in turn split into a new set of parallel processes, and so on. Importantly, Nondeterministic FSMs and Deterministic ones are completely equivalent in power, as it is always possible to convert a Nondeterministic FSM into a Deterministic one (Sipser, 2006).

1.1.2 Regular Languages

Let us first extend the notion of transition function of a FSM to operate on entire strings rather than single symbols.

Definition 1.2. We define the FSM *extended transition function* as a function $\hat{\delta} : Q, \mathbf{T}^* \rightarrow Q$, where Q is the set of states of the FSM, and T^* is known as the *Kleene star* of alphabet $\mathbf{T}$, i.e. the (infinite) set of all possible strings that can be formed by the finite concatenation of symbols from $\mathbf{T}$, which includes the empty string ϵ, the string containing no symbols. We define $\hat{\delta}$ by induction as follows:

Base case: $\hat{\delta}(q, \epsilon) = q$, that is, if the current state is q and no input is present, the state does not change.

Inductive step: let $w = w_1 d$ be a string where $d \in \mathbf{T}$ is the last symbol in w and $w_1 \in \mathbf{T}^*$ is the string obtained by considering all but the last symbol in w, then we define $\hat{\delta}$ such that

$$\hat{\delta}(q, w) = \delta\left(\hat{\delta}(q, w_1), d\right) \tag{1.2}$$

By the extended transition function, if we let w be some input string and q be some initial state (not necessarily the starting state), we can express the state reached by a FSM processing w and starting from q as $\hat{q} = \hat{\delta}(q, w)$. Through the extended transition function, we can now define the *language* of a FSM.

Definition 1.3. The language of a FSM $M_{\text{FSM}} = (Q, \mathbf{T}, q_0, F, \delta)$ is defined as

$$\mathcal{L}(M_{\text{FSM}}) = \{\ w \mid \hat{\delta}(q_0, w) \in F\}, \tag{1.3}$$

where $w \in \mathbf{T}^*$ is a string of symbols from the FSM input alphabet $\mathbf{T}$, and F is the set of accept states in the FSM, such that M_{FSM} is said to *accept* or *recognize* the language $\mathcal{L}(M_{\text{FSM}})$. If a language is accepted by a FSM, it is said to be a *Regular Language*.

In the Nondeterministic case, a FSM is said to accept a string if at least one of the sub-processes created by the FSM accepts it. In this case, the language of a Nondeterministic

FSM (NSFM) is defined as $\mathcal{L}(M_{\text{NFSM}}) = \{\ w \mid \hat{\delta}(O, w) \cap F \neq \varnothing\}$, where $\hat{\delta}$ is defined as before, but for sets of states rather than single ones.

1.1.3 Finite-State Transducers

Finite-State Transducers are an extension of Finite-State Machines, with the additional capability of producing an output symbol at every transition. This model is particularly relevant to the domain of Natural Language Processing, as it has been widely applied e.g. to morphology analysis, to tokenization and shallow parsing (Karttunen, 2000). We will later use this model to describe some forms of heteroclinic computation. We will only define Deterministic FST, as they are equivalent in power to their Nondeterministic counterpart.

Definition 1.4. A Deterministic Finite-State Transducer (DFST, or simply FST) can be formally defined as a 6-tuple $M_{\text{FSM}} = (Q, \mathbf{T}, \mathbf{\Gamma}, , q_0, F, \delta)$, where

1. Q is a finite set of machine states or *control* states,
2. $\mathbf{T}$ is a finite set of symbols known as input alphabet,
3. $\mathbf{\Gamma}$ is a second finite set of symbols known as output alphabet,
4. $q_0 \in Q$ is a distinguished state in Q known as the starting state, i.e. the state the FST is in at the beginning of its computation,
5. $F \subseteq Q$ is a set of final (or *accept*) states,
6. $\delta : Q \times \mathbf{T} \to Q \times \left(\mathbf{\Gamma} \cup \{\epsilon\}\right)$ is a transition function. Specifically, δ can be defined as follows:

$$\delta : (q_t, d_t) \mapsto (q_{t+1}, p_t), \tag{1.4}$$

where $q_t, q_{t+1} \in Q$ are states, $d_t \in \mathbf{T}$ is an input symbol, and $p_t \in \left(\Gamma \cup \{\epsilon\}\right)$ is an output symbol or the empty string ϵ.

A FST behaves similarly to a FSM, by consuming an input string symbol by symbol, and changing state at each step given its current state and the read symbol, as specified by the transition function δ. In addition to the usual repertory defined by FSMs, FSTs also produce output strings. In fact, the transition function δ of a FST specifies at each computation step t the production of an output symbol p_t by the machine.

As an extension of FSMs, FSTs can not only recognize the class of Regular Languages, but also, through the productions of output symbols, *generate* them.

1.1.4 Push-Down Automata

Finite-State Machines model computation with finite memory. Extending this class of models by endowing it with the additional capability of pushing and popping symbols from

a stack memory defines a more powerful[1] computing machine, the Push-Down Automaton (PDA).

Definition 1.5. A Deterministic Push-Down Automaton (DPDA, or simply PDA) can be formally defined as a 6-tuple $M_{\text{PDA}} = (Q, \mathbf{N}, \mathbf{T}, q_0, F, \delta)$, where:

1. Q is a finite set of machine, or *control*, states,
2. $\mathbf{N}$ is the a finite set of symbols known as the stack alphabet,
3. $\mathbf{T}$ is a second finite set of symbols known as the input alphabet,
4. $q_0 \in Q$ is a distinguished state from the set of possible machine states known as the starting state, i.e. the state the machine is in at the beginning of its computation,
5. $F \subseteq Q$ is a set of final, or *accept* states,
6. δ is a transition function.

At each computation step the PDA consumes an input symbol and, given the input symbol, the current symbol on the top of the stack and the current state, it transitions to a new state, pushing or popping a symbol to the top of the stack, as prescribed by its δ function. For a Deterministic PDA, the transition function can be defined as

$$\delta : Q \times \left(\mathbf{T} \cup \{\epsilon\}\right) \times \left(\mathbf{N} \cup \{\epsilon\}\right) \to Q \times \left(\mathbf{N} \cup \{\epsilon\}\right), \tag{1.5}$$

where ϵ is the empty string, whereas the transition function of a Nondeterministic PDA can be defined as

$$\delta : Q \times \left(\mathbf{T} \cup \{\epsilon\}\right) \times \left(\mathbf{N} \cup \{\epsilon\}\right) \to P\left(Q \times \left(\mathbf{N} \cup \{\epsilon\}\right)\right). \tag{1.6}$$

where $P\left(Q \times \left(\mathbf{N} \cup \{\epsilon\}\right)\right)$ is the powerset of the set of possible combinations of state and top-of-stack that can occur in the PDA. That is, similarly to what discussed in the case of Nondeterministic FSM, a Nondeterministic PDA can, at every computation step, span concurrent subprocesses each transitioning to a different configuration, where each subprocess can subsequently span new subprocesses and so on.

[1] The adjective "powerful" here is relative to the computational power of the model; if a model is computationally more powerful than another, then the class of languages the first model recognizes strictly contains that of the second model. Equivalently, the class of languages recognized by the first model is higher up in the Chomsky hierarchy (Chomsky, 1956).

Because of the additional capabilities of PDA with respect to FSMs, the notation for transitions is here slightly more complex. In particular for a PDA, if $s_{0_t} \dots s_{m_t} \in \mathbf{N}^*$ is the current content of the stack (where $\mathbf{N}^*$ is the set of all possible strings that can be obtained from concatenation of symbols in $\mathbf{N}$, included the empty string ϵ), with s_{0_t} as the current top-of-stack, and if q_t is the current state, q_{t+1} the new state, and d_{0_t} the current input symbol, transitions of the form

$$\delta : (q_t, d_{0_t}, s_{0_t}) \mapsto (q_{t+1}, \epsilon)$$

apply a pop operation, such that the new stack content becomes $s_{1_t} \dots s_{m_t}$. Push operations are instead applied by transitions of the form

$$\delta : (q_t, d_{0_t}, s_{0_t}) \mapsto (q_{t+1}, s_{0_{t+1}}),$$

where $s_{0_{t+1}}$ is the new top-of-stack, so that the updated stack contains the symbols $s_{0_{t+1}} s_{0_t} \dots s_{m_t}$. Finally, for transitions of the form

$$\delta : (q_t, \epsilon, s_{0_t}) \mapsto (q_{t+1}, \chi \in \{\epsilon\} \cup \mathbf{N}),$$

the PDA does not consume any input symbol (i.e. it does not access its input at all), but either pops its top-of-stack, if $\chi = \epsilon$, or pushes symbol χ, if $\chi \in \mathbf{N}$.[2] If the set of final states is empty, i.e. $F = \varnothing$, the PDA accepts its input when both the input tape and the stack are empty, and it is thus said to accept by *empty stack*. A NPDA accepts its input if one of the parallel processes it creates either ends its computation with an accept state, or with empty input and stack in the case where $F = \varnothing$.

Let us now give an example of a simple computation performed by a Deterministic PDA. Consider a PDA accepting binary strings which contain a certain number of `1`'s followed by the same number of `0`'s. The PDA can perform this computation by, say, pushing a special `x` symbol on its stack every time a `1` is read in input, and popping its top-of-stack every time a `0` is read instead. Such a DPDA can be defined as $M_{\text{eq}} = (Q, \mathbf{N}, \mathbf{T}, q_0, F, \delta)$ where $Q = \{q_0, q_{\text{count}}, q_{\text{accept}}\}$ is the set of states, $\mathbf{N} = \{\texttt{0}, \texttt{1}$ is the input alphabet, $\mathbf{T} = \{\texttt{x}, \texttt{\$}\}$ is the stack alphabet, with a special symbol `$` which we will use to allow the PDA to recognize

[2] Note that Deterministic PDA do not allow transitions of the form $\delta(q_t, \epsilon, \epsilon) \mapsto (q_{t+1}, \epsilon)$, known as *empty transitions*, as this would introduce for each configuration of state, input and stack, the possibility to perform an empty transition, breaking determinism if any other transition is defined.

when everything has been popped from the stack, $F = \{q_{\text{accept}}\}$ is the set of accept states containing only state q_{accept}, and where δ is defined as follows

$$\delta : \begin{cases} (q_0, \epsilon, \epsilon) \mapsto (q_{\text{eq}}, \texttt{\$}), \\ (q_{\text{count}}, \texttt{1}, \epsilon) \mapsto (q_{\text{!eq}}, \texttt{x}), \\ (q_{\text{count}}, \texttt{0}, \texttt{x}) \mapsto (q_{\text{count}}, \epsilon), \\ (q_{\text{count}}, \epsilon, \texttt{\$}) \mapsto (q_{\text{accept}}, \epsilon), \end{cases} . \tag{1.7}$$

At the very beginning of the computation, the machine pushes a `$` symbol as the top-of-stack, to function as a signal that the stack does not contain any further symbols. Subsequently, the machine starts reading its input string, by moving to state q_{count}. For each `1` it reads from its input string, the machine pushes an `x` on its stack; for each `0`, it pops an `x`. If all `x`'s are popped, the machine reads the `$` on the top of its stack, and transitions to state q_{accept}. If the input string has been completely read at this point, the machine completes its computation in q_{accept}, thus accepting the input string. Note that any string not matching the $\texttt{1}^n\texttt{0}^n$ pattern (where the exponent denotes repetition) will cause the machine to 'crash' because of undefined transitions. That is, consider for example an input string of `100`. The machine will

1. push the `$` symbol on the stack, moving from state q_0 to state q_{count},
2. read the `1`, pushing an `x` on the stack
3. read the `0`, popping an `x` from the top-of-stack
4. read the `$` symbol at the top-of-stack, transitioning to state q_{accept}.

At this point, the machine has yet another `0` left to read in its input, but the transition function is not defined for the triple $(q_{\text{accept}}, \texttt{0}, \epsilon)$ of state, input symbol and top-of-stack. The machine is thus said to reject the input string.

1.1.5 Context-Free Grammars and Languages

Definition 1.6. A Context-Free Grammar (CFG) can be formally defined as a 4-tuple $G_{CF} = (\mathbf{N}, \mathbf{T}, R, \mathtt{S})$, where

- $\mathbf{N}$ is a finite set of symbols known as *non-terminals*,
- $\mathbf{T}$ is a finite set of symbols known as *terminals*,
- $R \subset \mathbf{N} \times (\mathbf{N} \cup \mathbf{T})^*$ is a set of substitution rules (or productions, or rewriting rules) specifying how to substitute non-terminals with strings of terminals and non-terminals.
- $\mathtt{S}$ a distinguished start symbol.

Each rule (X, w) in R can be written as $X \to w$, with $X \in \mathbf{N}$ and $w \in (\mathbf{N} \cup \mathbf{T})^*$.

A Context-Free Language is a language generated by a CFG. A CFG generates a language by the recursive rewriting of its non-terminals starting from the distinguished starting non-terminal symbol $\mathtt{S}$. The rewriting steps involved in the generation of a string are called the *derivation* of the string.

For example, let's define a grammar G_{CFG} with $N = \{\mathtt{A}, \mathtt{B}\}$, $T = \{\mathtt{a}, \mathtt{b}\}$, starting symbol $\mathtt{S}$ and rules

$$\mathtt{S} \to \mathtt{A}$$

$$\mathtt{S} \to \mathtt{B}$$

$$\mathtt{S} \to \mathtt{a}$$

$$\mathtt{S} \to \mathtt{b}$$

$$\mathtt{S} \to \epsilon$$

$$\mathtt{A} \to \mathtt{aSa}$$

$$\mathtt{B} \to \mathtt{bSb}.$$

This grammar generates the language of palindromes on $\mathtt{a}$ and $\mathtt{b}$. An example derivation is

$$\mathtt{S} \to \mathtt{B} \to \mathtt{bSb} \to \mathtt{bAb} \to \mathtt{baSab} \to \mathtt{baaab}.$$

Importantly, it is always possible to construct, given any CFG, a NPDA recognizing its language, and vice versa. That is, NPDA recognize the class of languages generated by CFGs. Note that, contrary to Deterministic FSMs and Nondeterministic FSMs, Deterministic and Nondeterministic PDA differ in power. Specifically, DPDA can only recognize a subset of CFGs, known as Deterministic CFGs (DCFGs).

1.1.6 Top-Down Recognizers

Top-Down Recognizers (TDRs) are a subclass of NPDA that can simulate rule expansion to accept Context-Free Languages. They are particularly important as they are used prove the equivalence in power between CFGs and NPDA. Moreover, the deterministic restriction of TDRs has very important uses for parsing in real-world applications (Aho and Ullman, 1972).

Definition 1.7. Given a CFG $G_{\text{CFG}} = (\mathbf{N}, \mathbf{T}, R, \mathtt{S})$, a Top-Down Recognizer (TDR) is a NPDA $M_{\text{TDR}} = (\{q\}, \mathbf{N}, \mathbf{T}, q, F, \delta)$ with the following characteristics:

- If $(A \to s) \in R$ then $(q, s^r) \in \delta(q, \epsilon, A)$, where s^r denotes the reversed string s.
- $\delta(q, a, a) = (q, \epsilon)$ for all $a \in \mathbf{T}$

Informally, at each computation step a TDR checks its top-of-stack. If it is a nonterminal $A \in \mathbf{N}$ of the G_{CFG} grammar, the TDR nondeterministically applies all rule expansions for A, by popping A and pushing its reversed expansion s^r on the stack. If the top-of-stack is instead a terminal symbol $a \in \mathbf{T}$, the TDR checks whether it matches the current input symbol. If it matches, then the input symbols is consumed, and its match popped from the stack. If it does not match then the associated nondeterministic process rejects the input string. A TDR accepts its input if one of its processes ends its computation with an empty stack and input.

1.1.7 Turing Machines

To investigate the nature of computation and to probe its limits, Alan Turing devised an imaginary machine able to read and write symbols on an infinite memory support, simulating the action of a human computer (Turing, 1937). Turing imagined the action of a person carrying out a computation with pencil and paper. Stripping away the distracting details, Turing argues that the human computer performs a computation by going through a sequence of "mental states", while reading and rewriting a finite number of symbols on the paper at each step of the sequence.

Turing creates his machine in the image of a human computer: the mental states of the human computer become internal states of the machine; the pencil becomes a read-write head; the paper becomes a one-dimensional tape. At each step of a computation, the machine reads the current symbol under its read-write head and, depending on its internal state, rewrites the symbol, moves the head to the left or to the right, and updates its state (see 1.1 for a pictorial representation).

With this simple model, Turing is able to identify the atomic structure of Universal Computation: the widely held thesis known as the Church-Turing conjecture states that every function that can be computed by a human computer under no restraint on resources can be computed by a Turing Machine, and vice-versa.

While the conjecture has never been proven (and is actively challenged, for example, by proponents of hypercomputation, see Copeland, 2002) it has had and continues to have a profound influence on Science. Its consequences are central to many important theories in disparate fields such as Physics, Neuroscience, Biology, Cognitive Science and Philosophy.

Let us now introduce a formal definition of Turing Machine.

Definition 1.8. A Turing Machine (TM) is a 7-tuple $M_{\mathrm{TM}} = (Q, \mathbf{N}, \mathbf{T}, q_0, \sqcup, F, \delta)$, where:

1. Q is a finite set of machine states, or *control* states,
2. $\mathbf{N}$ is a finite set of tape symbols, always including the *blank* symbol $\sqcup$,
3. $\mathbf{T} \subset \mathbf{N} \setminus \{\sqcup\}$ is the input alphabet,
4. q_0 is a distinguished state from the set of states known as *starting* state, i.e. the state which the machine is in at the beginning of its computation,
5. $\sqcup$ is the blank symbol, the symbol contained in each empty cell of the TM tape,
6. $F \subset Q$ is a set of 'halting' states which, when reached, define the end of the computation performed by the TM,
7. δ is a partial transition function, the so-called machine table, that determines the dynamics of the machine. In particular, δ is defined as follows:

$$\delta : Q \times \mathbf{N} \to Q \times \mathbf{N} \times \{\mathcal{L}, \mathcal{R}\}, \tag{1.8}$$

where $\mathcal{L}$ and $\mathcal{R}$ denote a movement of the read/write head to the left or to the right, respectively.

A Turing Machine has read/write access to a memory support, a tape divided in cells, each containing one symbol. An empty cell contains the blank $\sqcup$ symbol. At every step of the computation, a Finite-State controller endowed with a read-write head follows the instructions encoded by the δ transition function. Given the current symbol under the head and its current state, the controller determines the transition to a new state, the writing of a new symbol in the current cell, and the movement of the head to the cell to the left ($\mathcal{L}$) or to the right ($\mathcal{L}$) of the current one. To simplify notation, it is possible to add a third movement to the repertoire of the Turing Machine, allowing for the read/write head to stay in place after rewriting. This *stay* operation will be denoted as $\mathcal{S}$. Note that any Turing Machine endowed with the additional $\mathcal{S}$ operator can be trivially implemented by one without, where every $\mathcal{S}$ can be substituted by two subsequent transitions in which the head is moved in two opposite directions, resulting in an equivalent null displacement.

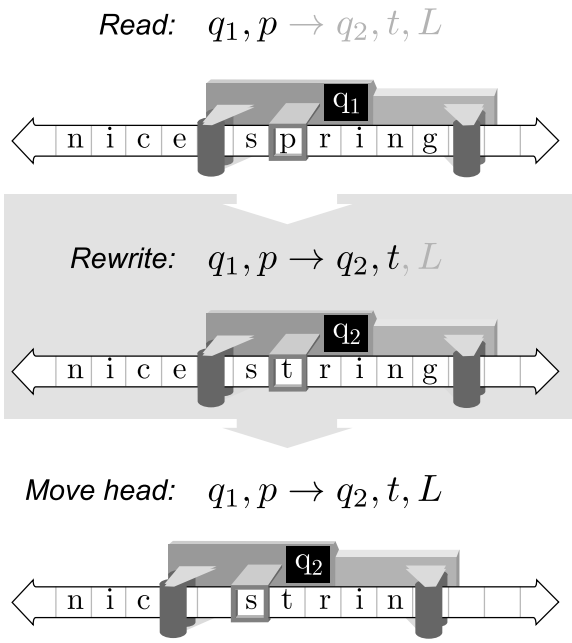


FIGURE 1.1: **A Turing Machine.** The Figure shows the three operations (read, rewrite, move head) that a Turing Machine performs at each computation step, as instructed by its transition table (not shown).

1.2 Information

> Information and computation are two complementary ideas in a similar way to continuum and a discrete set. In its turn continuum–discrete set dichotomy may be seen in a variety of disguises such as: time–space; wave–particle; geometry–arithmetic; interaction–algorithm; computation–information. Two elements in each pair presuppose each other, and are inseparably related to each other. [...] The field of Philosophy of Information is so closely interconnected with the Philosophy of Computation that it would be appropriate to call it Philosophy of Information and Computation, having in mind the dual character of information–computation.
>
> – Gordana Dodig-Crnkovic (2011)

The term "information" has not yet found a systematic definition, but rather determines a constellation of meanings that depend on the specific context in which the term is used (Burgin, 2003). Nevertheless, it is somewhat theoretically easier to give a general definition of information *processing*. In his *General Theory of Information*, Burgin (2010) distinguishes three basic ways in which information can be processed, as

- **Transmission**, i.e. changing the position of information in space,
- **Transformation**, i.e. changing the information itself, or its representation,

– **Storage**, i.e. changing its position in time.

Traditionally, computation is seen as any process that *transforms* information, or its representation, whereas information *transmission* is seen as a process of communication. Importantly, the *communication* of information has been very well characterized by the apparatus of classical Information Theory, founded by Claude E. Shannon in his 1948 seminal paper (for a complete introduction, see MacKay, 2003).

Nevertheless, as the limits of the classical Theory of Computation become more and more evident and interactive extension to the theory are being formulated, the boundary between computation and communication is increasingly being perceived as artificial, and efforts are being made to integrate the two concepts (Dodig-Crnkovic, 2011).

In this Section, we introduce measures of information (with the specific definition of 'eliminated uncertainty', discussed later) developed in the study of information communication from classical Information Theory. In later Chapters, we will use these quantities to explore the information transmission characteristics of a class of neuronal networks implementing formal systems.

1.2.1 Entropy

Entropy is a measure of information which characterizes the unpredictability of the outcomes of a certain process. Given an intuitive understanding of "information content" of some outcome, it can be argued that observing the outcome of a completely predictable process does not have any informative value: the outcome is already known in advance. On the other hand, observing the outcome of more unpredictable processes has a higher informative value, as the observer knowledge has to be updated upon observation. The more the outcome is unexpected, the more radical the updating has to be, the more informative the content of the outcome is.

A measure for the information content of an outcome, argues Shannon, should have some key desirable properties. First of all, one would like this measure to be a decreasing function of the probability of the outcome. In fact a rather unlikely outcome is less expected, less

predictable, and thus more informative. Second, the information content of two outcomes from two independent processes happening together should intuitively equal to the sum of the information contents of the single outcomes. An observer should not be able to acquire more information on a process by observing an additional, independent one.

Shannon demonstrated that the negative logarithm is the only function that satisfy these constraints . In fact $-\log p(y)$ decreases as the probability p of the outcome increases, and if the probability of two independent outcomes happening together is the product $p(y_1)p(y_2)$ (i.e. the two events are independent), then $-\log p(y_1)p(y_2) = -\log p(y_1) - \log p(y_2)$. The base of the logarithm can be chosen arbitrarily, but is traditionally set to 2, so that information can be related easily with results from systems relying on binary encodings (such as most digital computers).

The Entropy H of a process is defined as the average information of its outcomes, i.e.

$$H(Y) = -\sum_y p(y) \log_2 p(y), \tag{1.9}$$

where Y is the random variable representing the process, and y one of its outcomes.

1.2.2 Mutual Information

We introduce a second important measure of information, relating the outcomes of two processes X and Y in terms of how much information about one of the processes can be extracted by observing the other (where X and Y are again random variables). This is the Mutual Information $I(X;Y)$ between X and Y, defined as

$$I(X;Y) = H(X) - H(X|Y), \tag{1.10}$$

where $H(X)$ is here known as the *marginal entropy* of X and defined as in Equation 1.9, whereas $H(X|Y)$ is known as the *conditional entropy* of X given Y, defined as

$$\begin{aligned} H(X|Y) = & \sum_y p(y) H(X|Y=y) \\ = & -\sum_y p(y) \sum_x p(x|y) \log p(x|y), \end{aligned} \tag{1.11}$$

that is, the entropy of X given that $Y = y$ was observed, averaged over all possible y's.

We can think about the Mutual Information $I(X;Y)$ as the answer to the question: how much information can we obtain about X by observing Y? The Mutual Information between X and Y can thus be thought of as a drop in the uncertainty about X given that Y was observed. In fact, if $I(X;Y) = H(X) - H(X|Y) = 0$, then the entropy $H(X) = H(X|Y)$, meaning that observing Y does not modify the unpredictability of X (i.e. Y does not give any information about X). Mutual Information is instead at its maximum when $H(X|Y) = 0$, and thus $I(X;Y) = H(X)$; in this case, the unpredictability of X given that Y is observed becomes null, i.e. by knowing the outcome y of Y we are able to perfectly predict the outcome x of X, leading to a drop in uncertainty equal to $I(X;Y) = H(X)$.

Chapter 2

Dynamical Systems

In this Chapter, we give a short overview of some important objects in Dynamical Systems theory which we will use throughout the thesis. For a more complete introductory treatment of Dynamical Systems see ,for example, Glendinning (1994), Strogatz (2014), Wiggins (2003).

A dynamical system is characterized by a *state* or *phase space* X, a set containing points corresponding to all the possible states of the system, and by an evolution function that describes the time dependence of a point in this space.

At a coarse level, we can distinguish between dynamical systems evolving in discrete time, and those that evolve in continuous time.

Discrete-time dynamical systems. The evolution of a discrete-time dynamical system can be described by a map

$$x_{n+1} = \Phi(x_n), \tag{2.1}$$

where x_{n+1} and x_n are states in the system and $\Phi : X \to X$ a function mapping states to states.

Discrete-time piecewise affine-linear systems. This specific class of discrete-time systems will play a key role in the following Chapters. Discrete-time piecewise affine-linear systems are dynamical systems characterized by a set of affine-linear transformations in the form

$$x_{n+1} = \Lambda_\alpha x_n + A_\alpha \tag{2.2}$$

where $\alpha(x) \in \{1, \ldots, M\}$ is a switching rule depending on the current state of the system. That is, given the current state of the system the switching rule selects one transformation to apply to the state of the system from a set of affine-linear transformations.

Continuous-time dynamical systems. The class of continuous-time systems we will encounter in this work can be described by systems of ordinary differential equations of the form

$$\dot{x} = f(x), \tag{2.3}$$

where $x \in X$ is the state of the system, and where $\dot{x}$ denotes the rate of change of x over time as a function $f : X \to X$.

The dynamics of a discrete- or continuous-time system in the state space can be characterized by points and orbits with specific properties that define the long-term behaviour of the system:

Fixed points. A fixed point (or equilibrium) in the state space of a dynamical system is a point such that, i f the system is in that point, it stays there forever.

In the case of a discrete-time dynamical system as defined in Equation 2.1, this is a point x^* such that $x^* = \Phi(x^*)$. Here x^* is said to be a *stable* fixed point if all the eigenvalues of the Jacobian J of the linearization of $\Phi(x)$ at $x = x^*$ have magnitude smaller than 1; it is said to be *unstable* if all the eigenvalues of J have magnitude greater than 1; it is said to be a *saddle* point if at least one eigenvalue of J is greater than 1, at least one is smaller than 1, and no eigenvalue is equal to 1.

In the case of a continuous-time dynamical system as defined in Equation 2.3, a fixed point is a point x^* such that $f(x^*) = 0$. The point x^* is said to be a *stable* fixed point

if all the eigenvalues of the Jacobian J of the linearization of $f(x)$ at $x = x^*$ have negative real part; it is said to be *unstable* if all the eigenvalues of J have positive real part; it is said to be a *saddle* point if at least one eigenvalue of J has positive real part, and at least one has negative real part.[1]

Both in discrete- and continuous-time dynamical systems, a *stable* point is attractive, with nearby trajectories converging to it; an *unstable* point is repulsive, such that nearby trajectories diverge from it; a *saddle* point is instead attractive in some directions and repulsive in others.

Stable and unstable manifolds. The stable manifold $\mathcal{W}^s(x^*)$ of a fixed point x^* is a set of points in the phase space of a dynamical system which converge to x^* in forwards time, whereas the unstable manifold $\mathcal{W}^u(x^*)$ of a fixed point x^* is a set of points in the phase space of a system which converge to x^* in backwards time. That is,

$$\mathcal{W}^s = \left\{ x \;\middle|\; \lim_{t\to\infty} \big(f(x,t)\big) = x^* \right\} \tag{2.4}$$

$$\mathcal{W}^u = \left\{ x \;\middle|\; \lim_{t\to-\infty} \big(f(x,t)\big) = x^* \right\} \tag{2.5}$$

where f is the evolution function of the dynamical system.

Periodic orbit. A periodic orbit is a solution of a dynamical system that repeats itself over time.

For a discrete-time dynamical system defined as in Equation 2.1, a periodic orbit is defined as a set of points on the phase space $\big\{p_i = \Phi^i(p_0) \mid i = 1, \ldots, k-1\big\}$ such that $p_k = \Phi(p_0)$, where Φ^i denotes the composition of Φ with itself i times.

For a continuous-time dynamical system defined as in Equation 2.3, instead, a periodic orbit is defined as a set of points on the phase space $\big\{x(t) \mid t \in [0, T]\big\}$ such that $x(t) = x(t+T)$, where T is known as the *period* of the orbit.

Limit cycle. A periodic orbit is said to be a limit cycle if it is the limit set of some other trajectory of the dynamical system, i.e. the set of points to which the trajectory

[1] While some schools add an additional constraint to the definition excluding points with zero real part eigenvalues, we here choice to adhere to a more encompassing definition for the sake of simplicity.

converges after an infinite amount of time has passed (either forward of backward in time).

Of particular interest for this work, multiple saddle fixed points can sometimes be connected, thus defining specific graph-like structures in the phase space of a system:

Heteroclinic connections, cycles, networks. Two saddle points x_1^* and x_2^* are heteroclinically connected if at least part of the unstable manifold of x_1^* is contained in the stable manifold of x_2^*, where a stable manifold is defined as the set of points in phase space which converge in forwards time, and an unstable manifold as the set to which it converges in backwards time. When multiple saddles are connected to form a cycle, they are said to form a *heteroclinic cycle*. If multiple heteroclinic cycles are present in a system, they are said to be a *heteroclininic network* (see 2.1 for a visual depiction).

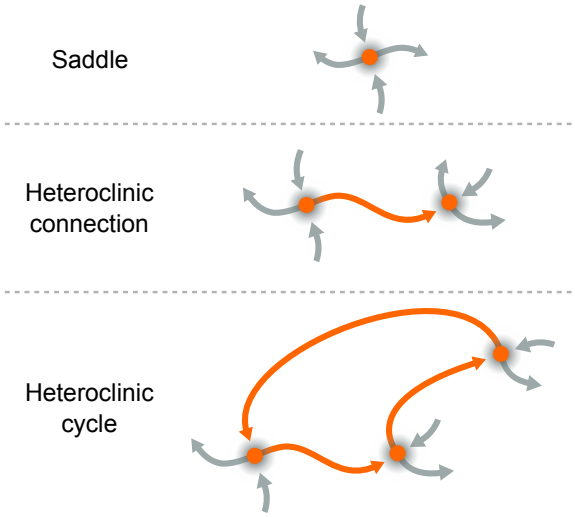


FIGURE 2.1: **Saddle point, heteroclinic connection and heteroclinic cycle**. The Figure shows a stylized depiction of a saddle point, a heteroclinic connection between two saddles, and a cycle between three saddles. The arrows pointing to a saddle, or away from it, represent its stable and unstable manifold, respectively.

Chapter 3

Symbolic Dynamics

In this Chapter, we introduce some key elements of the theory of Symbolic Dynamics to present the fundamental concepts that the latter Chapters will build upon. For a more comprehensive treatment of symbolic dynamics, we point the reader to the excellent introductory book by Lind and Marcus (1995).

We will first provide a few definitions for the basic abstractions in Symbolic Dynamics, to introduce the concepts and language essential to the description of the results of this thesis, specifically when discussing the dynamics of objects evolving in symbolic spaces. We will then move on to describe the Generalized Shift first introduced in Moore (1990), which we will extend in Chapter 4 to allow for the creation of Recurrent Neural Networks that transparently perform symbolic computation. In Section 3.2.1 we will present an adaptation of Moore's proof showing that the Generalized Shift is a Universal model of computation, and re-prove in Section 3.3 that the symbolic dynamics of a Generalized Shift can be mapped to point dynamics on a vector space.

3.1 Elements of Symbolic Dynamics

In this Section, we simply present a list of definitions functioning as an essential dictionary of the key objects in Symbolic Dynamics, and specifying the terminology used throughout the rest of this work when discussing dynamics (and computations) on strings of symbols.

Alphabet. An alphabet $\mathbf{A}$ is a finite set of symbols.

Dotted sequence. A dotted sequence $s = \ldots d_{-1} \,.\, d_0 \; d_1 \; \ldots,$ on some alphabet $\mathbf{A}$ is a sequence of symbols $(d_i)_{i\in\mathbb{Z}} \in \mathbf{A}^{\mathbb{Z}}$ where $d_i \in \mathbf{A}$, $\forall i \in \mathbb{Z}$.

One-sided infinite sequence. A one-sided infinite sequence $s \in \mathbf{A}^{\mathbb{N}}$ on some alphabet $\mathbf{A}$ is a sequence of symbols $(d_i)_{i\in\mathbb{N}}$ where $d_i \in \mathbf{A}$, $\forall i \in \mathbb{N}$.

Word (block). A word, or block, over an alphabet $\mathbf{A}$ is a finite sequence of symbols from $\mathbf{A}$. The length of a word d is denoted $|d|$ and is equal to the number of symbols in d. If $|d| = 0$ then d is the empty word, denoted as ϵ. A word of length k is called a k-word. The set containing all possible k-words over and alphabet $\mathbf{A}$ is denoted as $\mathbf{A}^k$. Let $s \in \mathbf{A}^{\mathbb{Z}}$, we denote the word with coordinates i, j in s (with $i, j \in \mathbb{Z}$) as $s_{[i,j]}$, and the word with coordinates $i, j-1$ (with $i < j-1$) as $s_{[i,j)}$. By extension, we will write $s_{[i,\infty)}$ to denote a right-infinite sequence and $s_{[-\infty,j]}$ to denote a left-infinite one.

Sub-word (sub-block). A sub-word or sub-block of some word $d = d_0 d_1 \ldots d_k$ is a word w such that $w = d_i d_{i+1} \ldots d_j$ with $0 \leq i \leq j \leq k$.

Full shift and shift space. The space $\mathbf{A}^{\mathbb{Z}}$ of dotted sequences is also known as full $\mathbf{A}$-shift. A subset $\mathbf{X}$ of the full shift $\mathbf{A}^{\mathbb{Z}}$ is a *shift space* (or just *shift*), which can be defined by specifying a collection of forbidden words $\mathcal{F}$, and denoted as $\mathbf{X}_{\mathcal{F}}$. The set of all possible n-words in a shift space $\mathbf{X}$ is denoted as $\mathcal{B}_n(\mathbf{X})$.

Shift of finite type. A shift of finite type is a shift space that can be described by a finite set of forbidden words $\mathcal{F}$.

Shift map. The shift map $\sigma : \mathbf{A}^{\mathbb{Z}} \to \mathbf{A}^{\mathbb{Z}}$ maps a dotted sequence $s \in \mathbf{A}^{\mathbb{Z}}$ to another sequence $\hat{s} = \sigma(s)$ by shifting all symbols in s to the right. Its inverse operation σ^{-1} shifts the symbols to the left.

Periodic dotted sequence. A dotted s sequence is periodic if $\sigma^n(s) = s$ for some $n \geq 1$, where n is said to be the *period* of x. The smallest n such that x is periodic with period n is the *least period* of x.

Language. The language of a shift space $\mathbf{X}$ is the collection $\mathcal{L}(\mathbf{X}) = \bigcup_{n=0}^{\infty} \mathcal{B}_n(\mathbf{X})$, where $\mathcal{B}_n(\mathbf{X})$ denotes the set of all n-words in $\mathbf{X}$.

Cylinder set. Given a shift space $\mathbf{X}$, a block $u \in \mathcal{B}(\mathbf{X})$, and an integer $k \in \mathbb{Z}$, a cylinder set $C_k(u)$ is defined as the set of all points in $\mathbf{X}$ containing the block u starting from position k, i.e. $C_k(u) = \{x \in \mathbf{X} \mid x_{[k,k+|u|)} = u\}$ where $|u|$ is the length of the block u.

N-th higher word code. The n-th higher word code is a bijective function β_n that maps n-words on an alphabet $\mathbf{A}$ to single symbols of a new alphabet $\mathbf{A}^{[n]}$. In other words, each n-word is now a single symbol in the new alphabet, such that

$$\left(\beta_n\left(x\right)\right)_i = x_{[i,i+n)} \tag{3.1}$$

maps the original dotted sequence to its n-th higher word representation.

3.2 Generalized Shifts

By extending the notion of shift map, it is possible to construct a more general map which can rewrite a finite number of symbols in a dotted sequence and shift it as a function of a finite number of places in the original dotted string. This map was introduced in Moore (1990, 1991) and named *Generalized Shift*.

Definition 3.1. A Generalized Shift (GS) on some alphabet $\mathbf{A}$ is a pair $M_{GS} = (\mathbf{A}^{\mathbb{Z}}, \Omega)$, with $\mathbf{A}^{\mathbb{Z}}$ being the space of dotted sequences on $\mathbf{A}$, and $\Omega : \mathbf{A}^{\mathbb{Z}} \to \mathbf{A}^{\mathbb{Z}}$ defined by

$$\Omega(s) = \sigma^{F(s)}\left(s \oplus G(s)\right) \tag{3.2}$$

with

$$\oplus : \mathbf{A}^{\mathbb{Z}} \times \left(\mathbf{A} \cup \{\chi\}\right)^{\mathbb{Z}} \to \mathbf{A}^{\mathbb{Z}} \tag{3.3}$$

$$F : \mathbf{A}^{\mathbb{Z}} \to \mathbb{Z} \tag{3.4}$$

$$G : \mathbf{A}^{\mathbb{Z}} \to \left(\mathbf{A} \cup \{\chi\}\right)^{\mathbb{Z}}. \tag{3.5}$$

Specifically,

1. G has the purpose of defining which symbols should be rewritten in the original dotted sequence s, by specifying the new symbols to be substituted to the old ones, and their positions. To do so, G produces a dotted sequence containing a distinguished χ symbol everywhere, with the exception of a finite number of symbols in the sequence, which position is specified by a fixed set of indices known as the *Domain of Effect* (or DoE) of the Generalized Shift. At these positions, the sequence produced by G contains the new symbols from $\mathbf{A}$ that the $\oplus$ operator will rewrite over the old ones in s. The identity of these new symbols depends on the contents of s, for a finite number of symbols in s specified by a set of indices known as the *Domain of Dependence* (DoD) of the Generalized Shift.

2. The $\oplus$ operator applies the substitution defined by the "blueprint" defined by G, i.e. $G(s) = (g_i)_{i\in\mathbb{Z}}$. Let us define $s^* = s \oplus G(s) = (s_i^*)_{i\in\mathbb{Z}}$, then for each index for which $g_i = \chi$, $\oplus$ leaves $s = (s_i)_{i\in\mathbb{Z}}$ unchanged, such that $s_i^* = s_i$. For all other indices $k \in$ DoE instead, i.e. for the indices in the Domain of Effect of the Generalized Shift, $s_k^* = g_k$. In this way, the combined action of the $\oplus$ operator and the function G applies the rewriting of a finite number of symbols in the original dotted sequence.

3. σ and F are respectively a shift map and a function defining how many positions and in which direction to shift. Specifically, the value of F depends on the contents of s in the indices specified by the Domain of Dependence of the Generalized Shift.

In the following Section we will present a concrete example of a Generalized Shift while discussing the simulation of Turing Machines with this model.

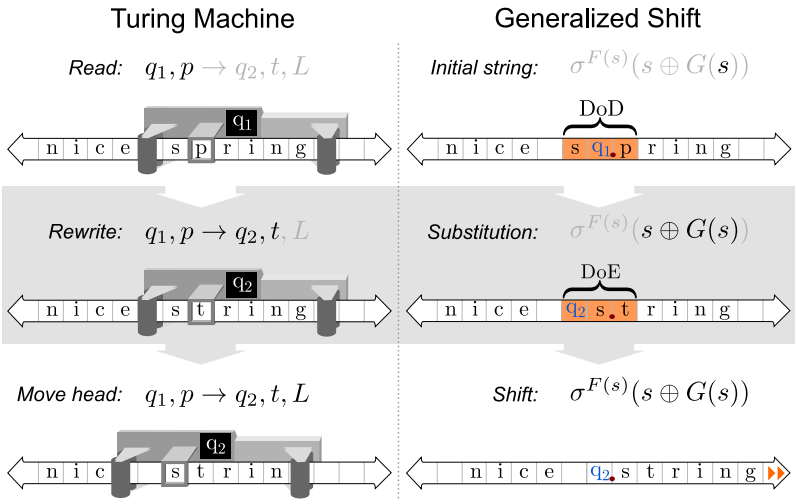


FIGURE 3.1: **Generalized Shift emulating a Turing Machine.** The Turing Machine starts in state q_1 with the tape contents being "nice spring" and the machine head being on "p". The toy computation that this machine is performing is the rewriting of symbols on the tape to obtain "nice string". This is achieved by first reading the tape cell with the symbol "p", then rewriting the same tape cell with symbol "t", and finally moving the control head to the left where the computation terminates in state q_2. The Generalized Shift mimics this computation by first applying a rewriting in the domain of effect (DoE) as a function of the contents of the domain of dependence (DoD) of the sequence, both set to be the set of indices $\{-2, -1, 0\}$ (highlighted in orange). Finally the Generalized Shift applies a shift map σ to the sequence, shifting it to the right in this case, and therefore obtaining a dotted sequence equivalent of the Turing Machine updated tape configuration.

3.2.1 Universality of Generalized Shifts

In what follows, we will give a short overview of the proof presented in Moore (1991) to show that the Generalized Shift can simulate any Turing Machine, and is thus a Universal model of computation.

The symbolic computation performed by a Turing Machine can be seen as a sequence of transitions between machine configurations. Each machine configuration is a complete description of the Turing Machine at that computation step, and consists of the current state of the controller, the current position of the read-write head on the tape, and the current content of the tape. The canonical description of a Turing Machine configuration is that of a triple (l_t, q, r_t), with q being a controller state, $l_t = \ldots d_{-2_t} d_{-1_t}$ being the contents of the tape to the left of the read-write head, and $r_t = d_{0_t} d_{1_t} d_{2_t} \ldots$ being the

contents to its right (and the d_{0_t} symbol currently under the head). Note how the position of the head-write head on the tape is implicitly encoded in this representation.

Equivalently, the configuration c of a TM $M_{\text{TM}} = (Q, \mathbf{N}, \mathbf{T}, q_0, \sqcup, F, \delta)$ can be represented as a dotted sequence $s \in (Q \cup \mathbf{N} \cup \mathbf{T})^{\mathbb{Z}}$ as follows

$$\Sigma_s(c) = \underbrace{\ldots\ d_{-2_t}\ d_{-1_t}}_{l_t}\ q_t\ .\ \underbrace{d_{0_t}\ d_{1_t}\ d_{2_t}\ \ldots}_{r_t}, \tag{3.6}$$

where $l_t = \ldots\ d_{-2_t}\ d_{-1_t}$ and $r_t = d_{0_t}\ d_{1_t}\ d_{2_t}\ \ldots$ are respectively the part of the tape to the left and the right of the read-write head (d_{0_t} being the symbol currently under the head), and q_t being the state of the machine controller. The central dot specifies the current position of the read-write head (i.e. d_{0_t}, the symbol to its right).

For a Generalized Shift $\Omega(s) = \sigma^{F(s)}\left(s \oplus G\left(s\right)\right)$ to emulate a Turing Machine, it is sufficient to set both the DoD and DoE of the Generalized Shift to coincide with the indices $\{-2, -1, 0\}$, thus specifying the sub-word $d_{i_{-1}}\ q_t\ .\ d_{i_0}$, and to construct F and G appropriately (see Figure 3.1 for a visual depiction). Specifically, given $\delta : (q_t, d_{0_t}) \mapsto (q_{t+1}, \hat{d}_{0_t}, m)$, i.e. the TM transition function defined in 1.8, F and G can be defined as

$$\begin{aligned}
G : s = \ \ldots\ d_{-1_t}\ q_t\ .\ d_{0_t}\ \ldots\ &\mapsto \begin{cases} \ldots\ (\chi)\ d_{-1_t}\ \hat{d}_{0_t}\ \ .\ q_{t+1}\ (\chi)\ \ldots & \text{if } m = \mathcal{R} \\ \ldots\ (\chi)\ q_{t+1}\ d_{-1_t}\ .\ \hat{d}_{0_t}\ \ (\chi)\ \ldots & \text{if } m = \mathcal{L} \end{cases} \\
F : s = \ \ldots\ d_{-1_t}\ q_t\ .\ d_{0_t}\ \ldots\ &\mapsto \begin{cases} -1 & \text{if } m = \mathcal{R} \\ +1 & \text{if } m = \mathcal{L} \end{cases}
\end{aligned} \tag{3.7}$$

for all $d_{-1_t} \in \mathbf{N}$.

We will now introduce an example to better show how Generalized Shifts can support the real-time simulation of Turing Machines. Consider a dotted sequence "$\mathtt{w}q_0\mathtt{.ord}$" representing the configuration of a TM with $\delta : (q_0, \mathtt{o}) \mapsto (q_1, \mathtt{a}, \mathcal{R})$ and $\delta : (q_1, \mathtt{r}) \mapsto (q_1, \mathtt{n}, \mathcal{L})$. Given this transition function, running the TM for two time steps starting from the "$\mathtt{w}q_0\mathtt{.ord}$" configuration would yield

$$c_1 = \mathtt{w}q_0\mathtt{.ord}, \qquad c_2 = \mathtt{wa}q_1\mathtt{.rd}, \qquad c_3 = \mathtt{w}q_0\mathtt{.and}$$

where c_t is the dotted sequence representing the machine configuration at time t. Constructing a GS Ω_{ex} as defined in 3.7, and applying it to "$\texttt{w}q_0\texttt{.ord}$" two times yields

$$
\begin{aligned}
\Omega^2_{\text{ex}}(c_1) &= \Omega^2_{\text{ex}}(\texttt{w}q_0\texttt{.ord}) \\
&= \Omega_{\text{ex}}\left(\sigma^{F(\texttt{w}q_0\texttt{.ord})}\left(\texttt{w}q_0\texttt{.ord} \oplus G\left(\texttt{w}q_0\texttt{.ord}\right)\right)\right) \\
&= \Omega_{\text{ex}}\left(\sigma^{-1}\left(\texttt{w}q_0\texttt{.ord} \oplus \texttt{wa.}q_1\right)\right) \\
&= \Omega_{\text{ex}}\left(\sigma^{-1}\left(\texttt{wa.}q_1\texttt{rd}\right)\right) \\
&= \Omega_{\text{ex}}(\texttt{wa}q_1\texttt{.rd}) \\
= \quad \Omega_{\text{ex}}(c_2) &= \sigma^{F(\texttt{wa}q_1\texttt{.rd})}\left(\texttt{wa}q_1\texttt{.rd} \oplus G\left(\texttt{wa}q_1\texttt{.rd}\right)\right) \\
&= \sigma^{+1}\left(\texttt{wa}q_1\texttt{.rd} \oplus q_0\texttt{a.n}\right) \qquad (3.8) \\
&= \sigma^{+1}\left(\texttt{w}q_0\texttt{a.nd}\right) \qquad (3.9) \\
&= \texttt{w}q_0\texttt{.and} \\
= \qquad c_3
\end{aligned}
$$

where the DoD of the input string to the VS has been highlighted for clarity. Dotted sequences representing TM machine configurations always contain a machine state at index -1, as shown in Equation 3.6. This is the reason why the DoD and DoE of GSs simulating Turing Machines must span not only the indices for the current state and symbol under the read-write head (respectively indices -1 and 0), but also – contrary to intuition – the index for the symbol to the left of the read-write head, i.e index -2. In fact, for a GS to simulate a TM transition $\delta(q_t, d_{0_t}) = (q_{t+1}, \hat{d}_{0_t}, \mathcal{L})$ where the read-write head is shifted to the left after rewriting of the current symbol, the GS rewriting $s \oplus G(s)$ must leave the current state displaced one place to the left (i.e. occupying index -2), such that the subsequent shift $\sigma^{+1}\left(s \oplus G(s)\right)$ can re-place it in its reserved index -1 (see lines 3.8 and 3.9 in the previous example).

3.3 Dynamical Systems from Symbolic Dynamics

In this Section, we discuss how the theory of Symbolic Dynamics can be used to construct dynamical systems that perform symbolic computation in vectorial spaces. We first introduce the key tool allowing the passage between symbolic shifts and vectorial spaces, i.e. the Gödel encoding (or Gödelization). We then show that the representation of Generalized Shifts on a vectorial space through Gödelization defines piecewise affine-linear systems on the unit square, known as Nonlinear Dynamical Automata.

3.3.1 Gödel encodings and the symbol plane

A Gödel encoding Gödel (1931) is a type of fractal encoding that maps one-sided infinite sequences to real numbers and thus allows the mapping of symbolic spaces to vectorial spaces. In this way, it becomes possible to describe classical symbolic models of computation as discrete time dynamical systems evolving on real vector spaces.

In what follows, the encoding process is discussed. See Figure 3.2 for a graphical representation.

Definition 3.2. A Gödel encoding $\psi : \mathbf{A}^{\mathbb{N}} \to [0,1] \subset \mathbb{R}$ can be defined as

$$\psi(s) := \sum_{k=1}^{\infty} \gamma(r_k) g^{-k}, \tag{3.10}$$

where s is a right-infinite sequence from $\mathbf{A}^{\mathbb{N}}$, the space of right-infinite sequences over some alphabet $\mathbf{A}$, r_k is the k-th symbol in s, $g = |\mathbf{A}|$ is the number of symbols in $\mathbf{A}$, and $\gamma : \mathbf{A} \to \mathbb{N}$ is a one-to-one function enumerating the symbols in $\mathbf{A}$ with $\max_r \gamma(r) < g$.

It is possible to Gödelize dotted sequences $w_l.w_r$ by defining two Gödel encodings ψ_x and ψ_y and applying them to the two right-infinite sequences w_l^{R} and w_r obtained by splitting the original dotted sequence at the dot and reversing its left constituent. The pair $\left(\psi_x\left(w_l^{\mathrm{R}}\right), \psi_y\left(w_r\right)\right)$ with $w_l^{\mathrm{R}}, w_r \in \mathbf{A}^{\mathbb{N}}$ induces a bi-dimensional representation of dotted sequences on the unit square $[0,1]^2 \subset \mathbb{R}^2$, known as symbol plane or symbologram (See Figure 3.3).

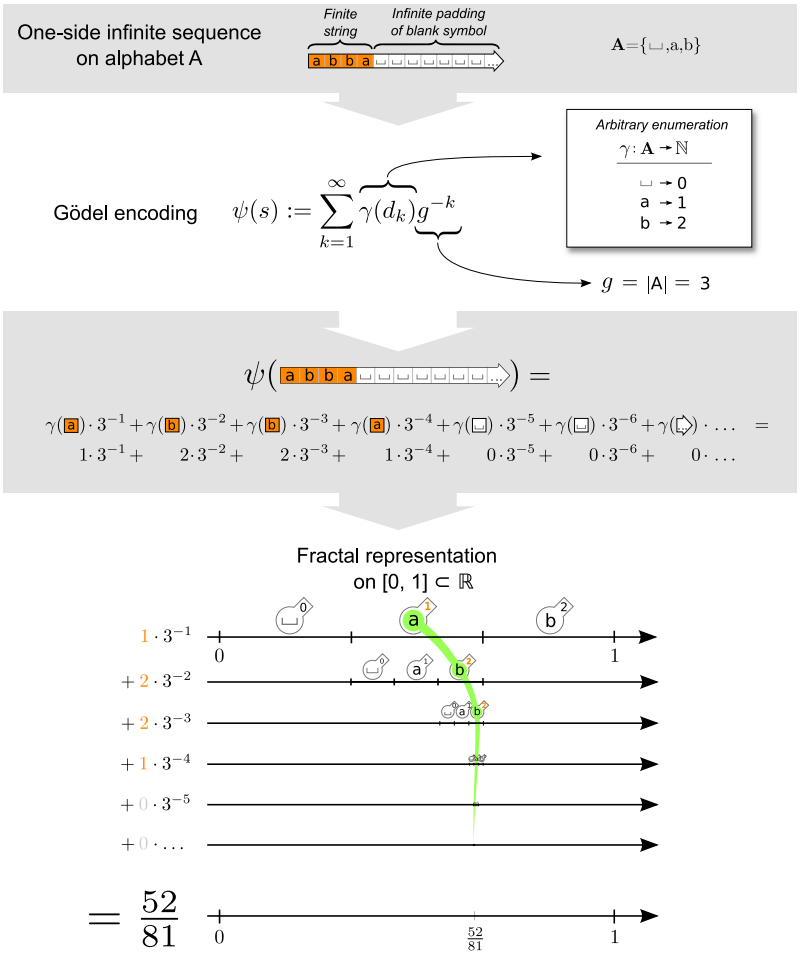


FIGURE 3.2: **Three representations of the Gödel Encoding of a sequence.** The first one is just the definition of the Gödel encoding, with details on the specific choice of the enumerating γ function and the definition of the g constant, given the alphabet $\mathbf{A}$ from which the sequence takes its symbols. The second one is an expansion of the series in the definition. The third one visually conveys the fractal and convergent nature of the series, highlighting the relation between numbers and symbols by the use of the colour orange.

Some important limits of the Gödelization approach in mapping symbolic to vectorial spaces are discussed in Section 4.6.

3.3.2 Shifts and rewritings as affine-linear transformations

In what follows, we introduce a *push* $\odot$ and *pop* $\ominus$ operator on one-sided infinite sequences, and show that rewritings and shifts by a Generalized Shift on dotted sequences can be mapped to push and pop operations on their one-sided infinite constituents. We will then show that the Gödelization of a one-sided infinite sequence $\hat{s}$ resulting from push

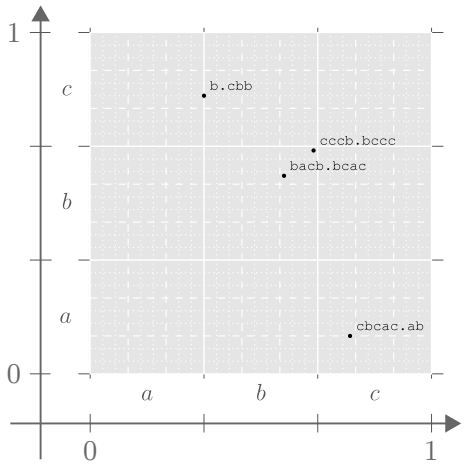


FIGURE 3.3: **Example symbolgram representation** for gödelized dotted sequences over an alphabet $\mathbf{A} = \{a, b, c\}$. Four encoded dotted sequences are shown on the unit square. Furthermore, a grid is imposed over the unit square, such that each cell specified by the n-th subdivision level contains dotted sequences agreeing on the first n symbols to the left and to the right of the dot. This is done to highlight the fractal nature of the symbolgram.

and pop operations on an original sequence s is equivalent to the application of a linear transformation on the Gödelization of the original sequence. In this way, we can prove that rewritings and shifts by a Generalized Shift on dotted sequences are equivalent, on the symbolgram, to linear transformations applied to their Gödelized one-sided constituents.

Definition 3.3. The push operator $\odot : \mathbf{A}^* \times \mathbf{A}^{\mathbb{N}} \to \mathbf{A}^*$ is defined as

$$s \odot b = bs, \tag{3.11}$$

such that $s \odot b$ adds the contents of a word $b \in \mathbf{A}^*$ to the beginning of $s \in \mathbf{A}^{\mathbb{N}}$. The pop operator $\ominus : \mathbb{N} \times \mathbf{A}^{\mathbb{N}} \to \mathbf{A}^{\mathbb{N}}$ is instead defined as

$$\ominus^p s = s_{[i \geq p]}, \tag{3.12}$$

such that $\ominus^p s$ removes the first p symbols in s.

It is very easy to show that shifting a dotted sequence $s = \ldots\ d_{-2}\ d_{-1}\ .\ d_0\ d_1\ \ldots$ is equivalent to applying push and pop operations on its one-sided constituents $w_l =$

$\ldots\ d_{-2}\ d_{-1}$ and $w_r = d_0\ d_1\ \ldots$, as follows

$$\begin{aligned}\sigma^{-1}(\ldots\ d_{-2}\ d_{-1}\ .\ d_0\ d_1\ \ldots) &= (w_l^{\mathrm{R}} \odot d_0)^{\mathrm{R}}.(\ominus^1 w_r)\\ &= \ldots\ d_{-1}\ d_0\ .\ d_1\ d_2\ \ldots\ ,\end{aligned}$$

and

$$\begin{aligned}\sigma^{1}(\ldots\ d_{-2}\ d_{-1}\ .\ d_0\ d_1\ \ldots) &= (\ominus^1 w_l^{\mathrm{R}})^{\mathrm{R}}.(w_r \odot d_{-1})\\ &= \ldots d_{-3}\ d_{-2}\ .\ d_{-1}\ d_0\ \ldots,\end{aligned}$$

where x^{R} denotes the x sequence in reverse order.

For what concerns the GS rewriting, instead, let us consider a GS with DoE $= D \subset \mathbb{Z}$ with $|D| = n \in \mathbb{N}^+$. As the DoE is finite, it contains a minimum index $k_{\min} = \min(\text{DoE})$ and a maximum index $k_{\max} = \max(\text{DoE})$ (the two can coincide). This means that symbols in s with indices $i > k_{\max}$ or $i < k_{\min}$ will never be rewritten by the GS. We thus have the three following possibilities:

- $\underline{k_{\min} < 0, k_{\max} < 0}$. Let us define $s = w_l\ u\ .\ w_r$, such that $w_l = s_{[-\infty,k_{\min})}$, $u = s_{[k_{\min},0)}$ and $w_r = s_{[0,+\infty]}$. Additionally, let us define $G(s) = \chi_l\ \hat{u}\ .\ \chi_r$, with $\hat{u} = \big(s \oplus G(s)\big)_{[k_{\min},0]}$, and χ_l, χ_r being respectively an infinite left and right padding of the χ symbol. Note that here $s \oplus G(s) = w_l\ \hat{u}\ .\ w_r$, as the GS rewriting does not affect w_l and w_r, because they are outside its DoD. It is possible in this case to map the GS rewriting $s \oplus G(s)$ to pop and push operations on one-sided infinite sequences as follows:

$$\begin{aligned}w_l\ u\ .\ w_r \oplus \chi_l\ \hat{u}.\ \chi_r &= \big((\ominus^{|u|} u^{\mathrm{R}} w_l{}^{\mathrm{R}}) \odot \hat{u}^{\mathrm{R}}\big)^{\mathrm{R}}.w_r\\ &= \big(w_l{}^{\mathrm{R}} \odot \hat{u}^{\mathrm{R}}\big)^{\mathrm{R}}.w_r\\ &= w_l\ \hat{u}\ .\ w_r,\end{aligned}$$

- $\underline{k_{\min} \geq 0, k_{\max} \geq 0}$. Let us define $s = w_l\ .\ u\ w_r$, such that $w_l = s_{[-\infty,-1]}$, $u = s_{[0,k_{\max}]}$ and $w_r = s_{(k_{\max},+\infty]}$. Additionally, let us define $G(s) = \chi_l\ .\ \hat{u}\ \chi_r$, with $\hat{u} = \big(s \oplus G(s)\big)_{[0,k_{\max}]}$, and χ_l, χ_r being respectively an infinite left and right padding of the χ symbol. Here $s \oplus G(s) = w_l\ .\ \hat{u}\ w_r$. The GS rewriting $s \oplus G(s)$ is thus

mapped to pop and push operations as follows:

$$\begin{aligned} w_l \,.\, u\; w_r \oplus \chi_l \,.\, \hat{u}\; \chi_r &= w_l \,.\, (w_r \odot \hat{u}) \\ &= w_l \,.\, \hat{u}\; w_r, \end{aligned}$$

– $\underline{k_{\min} < 0, k_{\max} \geq 0}$. Let us define $s = w_l\; u \;.\; v\; w_r$, such that $w_l = s_{[-\infty, k_{\min})}$, $u = s_{[k_{\min}, 0)}$, $v = s_{[0, k_{\max}]}$ and $w_r = s_{(k_{\max}, +\infty]}$. Additionally, let us define $G(s) = \chi_l\; \hat{u} \;.\; \hat{v}\; \chi_r$, with $\hat{u} = \big(s \oplus G(s)\big)_{[k_{\min}, 0)}$, $\hat{v} = \big(s \oplus G(s)\big)_{[0, k_{\max}]}$, and χ_l, χ_r being an infinite left and right padding of the χ symbol. Here $s \oplus G(s) = w_l\; \hat{u} \;.\; \hat{v}\; w_r$. The GS rewriting $s \oplus G(s)$ is here mapped to pop and push operations as follows:

$$\begin{aligned} w_l\; u \;.\; v\; w_r \oplus \chi_l\; \hat{u} \;.\; \hat{v}\; \chi_r &= \big((\ominus^{|u|} u^{\mathrm{R}} {w_l}^{\mathrm{R}}) \odot \hat{u}^{\mathrm{R}}\big)^{\mathrm{R}} . \big((\ominus^{|v|} v w_r) \odot \hat{v}\big) \\ &= \big({w_l}^{\mathrm{R}} \odot \hat{u}^{\mathrm{R}}\big)^{\mathrm{R}} . (w_r \odot \hat{v}) \\ &= w_l\; \hat{u} \;.\; \hat{v}\; w_r. \end{aligned}$$

Shifts and rewritings by a GS on dotted sequences can thus be mapped to pop and push operations on their one-sided infinite constituents. We will now show that the Gödelization of a one-sided sequence $\hat{s}$ resulting from pop and push operations on some original sequence s is equivalent to the application of an affine-linear transformation (a transformation of the form $\lambda x + a$, with $\lambda, a \in \mathbb{R}$) to the Gödelization of the original sequence $x = \psi(s)$. In fact, given a sequence $s = d_0\; d_1\; d_2\; \ldots$ and its Gödelization $\psi\,(s) = \gamma(d_1) g^{-1} + \gamma(d_2) g^{-2} + \gamma(d_3) g^{-3} + \ldots$, for a pop operation we obtain

$$\begin{aligned} \psi(\ominus^p s) &= \gamma(d_{p+1}) g^{-1} + \gamma(d_{p+2}) g^{-2} + \gamma(d_{p+3}) g^{-3} + \ldots \\ &= g^p \cdot \psi(s) - \sum_{i=1}^{p} \gamma(d_i) g^{p-i}, \end{aligned} \tag{3.13}$$

where the parameters of the affine-linear transformation are $\lambda = g^p$ and $a = -\sum_{i=1}^{p} \gamma(d_i) g^{p-i}$. For a push operation, instead, we have

$$\begin{aligned} \psi(s \odot b) &= \gamma(b_1) g^{-1} + \ldots + \gamma(b_r) g^{-r} + \\ &\quad \gamma(d_1) g^{-(r+1)} + \gamma(d_2) g^{-(r+2)} + \ldots \\ &= g^{-r} \cdot \psi(s) + \sum_{i=1}^{r} \gamma(b_i) g^{-i}, \end{aligned} \tag{3.14}$$

where the parameters of the affine-linear transformation are $\lambda = g^{-r}$ and $a = \sum_{i=1}^{r} \gamma(b_i) g^{-i}$.

In this way we can represent dotted sequences on the unit square by defining an appropriate pair of Gödelizations, and GS on dotted sequences as bidimensional affine-linear transformations, with the parameters of the transformations only depending on the symbols of the dotted sequence in the DoD of the GS. All dotted sequences sharing the same DoD symbols are thus associated with the same transformation, so that the symbologram representation of GSs naturally leads to piecewise affine-linear systems on the unit square, known as Nonlinear Dynamical Automata (NDA).

3.3.3 Nonlinear Dynamical Automata

We now present a formal definition of Nonlinear Dynamical Automata (beim Graben et al., 2008, 2004, Moore, 1991, Tabor, 2000, Tabor et al., 2013) to support discussion in the next Chapters.

Definition 3.4. A Nonlinear Dynamical Automaton (NDA) can be defined as a triple $M_{\text{NDA}} = (X, P, \Phi)$, where the pair (X, Φ) defines a discrete-time dynamical system, and such that

- P is a rectangular partition of the unit square, that is

$$P = \{D^{i,j} \subset X |\ 1 \leq i \leq m,\ 1 \leq j \leq n,\ m, n \in \mathbb{N}\}, \tag{3.15}$$

  so that each cell is defined as $D^{i,j} = I_i \times J_j$, with $I_i, J_j \subset [0, 1]$ being real intervals for each bi-index (i, j), with $D^{i,j} \cap D^{k,l} = \emptyset$ if $(i, j) \neq (k, l)$, and $\bigcup_{i,j} D^{i,j} = X$,

- $X = [0, 1]^2 \subset \mathbb{R}^2$, is the phase space of the (X, Φ) dynamical system, X corresponding in this case to the unit square,

- the flow $\Phi : X \to X$ is a piecewise affine-linear map such that $\Phi_{|D^{i,j}} := \Phi^{i,j}$, where $\Phi^{i,j}$ is defined as:

$$\Phi^{i,j}(\mathbf{x}) = \begin{pmatrix} a_x^{i,j} \\ a_y^{i,j} \end{pmatrix} + \begin{pmatrix} \lambda_x^{i,j} & 0 \\ 0 & \lambda_y^{i,j} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.16}$$

We also explicitly define a switching rule $\Theta : [0, 1]^2 \to \mathbb{N}^2$ for the piecewise affine-linear system (X, Φ), selecting the appropriate branch given the current state (x, y) of the system, such that

$$(x, y) \in D^{i,j} \quad \Rightarrow \quad \Theta(x, y) = (i, j) \quad \Rightarrow \quad \Phi(x, y) = \Phi^{i,j}(x, y). \tag{3.17}$$

The switching rule is implied in the original formulation of NDA, but we prefer to make it explicit to facilitate the discussion of the system later on.

We can define a mapping between GSs and NDA following the methods outlined in Sections 3.3.1 and 3.3.2. Specifically, the symbologram representation of dotted sequences can be partitioned in $I \times J$ cells such that each cell contains all Gödelized dotted sequences which have the same symbols in the DoD of the GS (see Figure 3.3 for a pictorial representation). As each cell in the rectangular partition P corresponds to a specific DoD content, it is also associated with a specific action of the GS, which can be mapped to pop and push operations on the left and right components of a dotted sequence, and thus to a

bi-dimensional affine-linear transformation which parameters can be derived by following Section 3.3.2. As we have shown in Section 3.2.1, GSs can simulate any Turing Machine by setting the DoD = DoE = $\{-1, 0, 1\}$ and appropriately constructing the G and F functions. Given that the symbolic dynamics of GSs can be mapped to the NDA vectorial dynamics on the unit square, NDA can thus also simulate any Turing Machine in real time. TMs can thus be represented as NDA through the Gödelization of their simulating GSs.

Chapter 4

Transparent symbolic computation in Neural Networks

The field of computational neuroscience is devoted to understanding how networks of biological neurons are able to process information and thus perform computation. In spite of that, there is a stark disconnect between our descriptions of symbolic computation, traditionally formulated in terms of algorithms, automata and formal languages, and descriptions of neuronal computation, which are instead usually formulated in terms of dynamical evolutions on state spaces. Early attempts to bridge these two levels of description can be traced back to pioneering work by McCulloch and Pitts (1943), who modelled neurons as binary logic gates. Networks of McCulloch-Pitt neurons have since been proven to be equivalent in power to Finite-State Machines thanks to seminal work by Kleene (1956) and Minsky (1967). More recently, Siegelmann and Sontag (1991, 1995) showed that a constructive mapping can be defined between Turing Machines and Recurrent Neural Networks (RNNs) with rational weights and ramp activation functions. Moreover, they proved that, given an appropriate adjacency matrix, a network of 886 ramp units can simulate a Universal Turing Machine, thus showing that RNNs are, in fact, Universal models of computation. Stemming from this work, Cabessa and Siegelmann (2012), Cabessa and Villa (2012) showed that RNNs can also support the simulation of interactive Turing Machines, further extending the boundaries of RNN information processing to the realm of interactive

computation (Wegner, 1998). Furthermore, these authors have also shown that RNNs can theoretically support forms of computation which are fundamentally outside what can be modelled in terms of Turing Machines (Cabessa and Villa, 2013, Siegelmann, 1995).

In this Chapter, we set ourselves to lay a new stone on the path traced by these pioneers. Specifically, we first extend work done by Moore (1990, 1991) on Generalized Shifts and Nonlinear Dynamical Automata (GSs and NDA, see Chapter 3), by presenting a new shift map, the *Versatile Shift* (VS, discussed in Section 4.1), that supports the parsimonious real-time simulation of a range of symbolic computation models (including but not restricted to TMs, see Section 4.2), and show that its Gödelization defines NDA evolving on the unit square (Section 4.3). Secondly, we present a constructive mapping between NDA and RNNs (in Section 4.4), describing a transparent, modular and parsimonious architecture for symbolic computation in Neural Networks.

The key novelty of this contribution stems from the relation between the network architecture and the structure of NDA. Crucially, similarly to how symbolic models of computation (or any formal system) distinguish between data, operations on data, and their controlled application, NDA also preserve, in their formulation, the distinction between these three. By constructing RNNs modularly from the NDA components, our architecture keeps this distinction and thus implements symbolic computation in a transparent and straightforward manner, as we will discuss in Section 4.4. To demonstrate the power of our framework, we will present three examples (Section 4.5). With these, we show that the granular modularity and transparency of the architecture allows for the constructive mapping of Interactive Automata Networks to RNNs, and opens the possibility of correlational studies with electrophysiological data.

In Figure 4.1, we present a commutativity diagram tracing the various steps involved in our formulation (on the left), and which Section discuss them (on the right). Starting from an automaton evolving on its configuration space, we derive a Versatile Shift evolving in the space of dotted sequences, by mapping configurations to sequences through a map Σ_s, and the automaton transition function δ to the Versatile Shift update function Ω through a map Σ_δ. The Versatile Shift object is discussed in Section 4.1, whereas the mapping

between various automata dynamics to Versatile Shift dynamics through the Σ_s and Σ_δ maps is discussed in Section 4.2. From the Versatile Shift, we then map dotted sequences to points on a bi-dimensional vector space through a Gödel encoding ψ, and the Versatile Shift Ω update rule to the piecewise affine-linear dynamics Φ of a NDA through a map Ψ. In this way, the Versatile Shift dynamics on the symbolic space of dotted sequences can be mapped to point dynamics on the unit square. The maps ψ and Ψ are presented in Section 4.3. Finally, we show how the dynamics Ω of a NDA can be mapped to the dynamics ζ of a Recurrent Neural Network endowed with a specific structure through a map ρ. The maps ζ and ρ are discussed in Section 4.4.

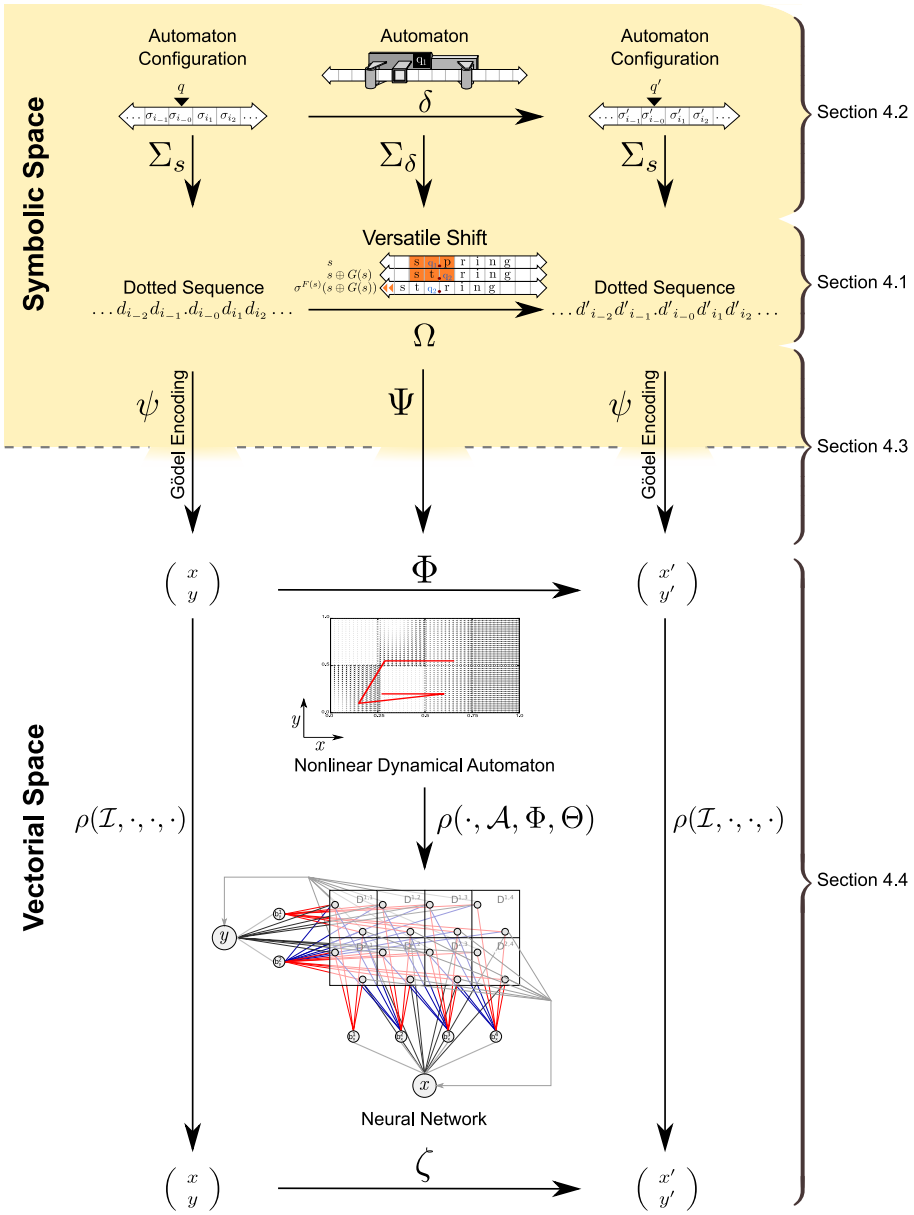


FIGURE 4.1: **Complete commutativity diagram for mapping of models of computation to the proposed RNN architecture**. In the Figure, we show how an automaton can be mapped to a RNN simulating its computation in real-time. The automaton configurations are first represented as dotted sequences through a Σ_s map, and the automaton itself mapped to a Versatile Shift (see Section 4.1) on dotted sequences through a Σ_δ map (the maps Σ_s and Σ_δ are discussed in Section 4.2 for FSMs, PDA, TDRs and TMs). The dotted sequences can then be represented as points on a bi-dimensional vector space through a ψ Gödelization, and the Versatile Shift as a piecewise affine-linear system through a map Ψ, obtaining a Nonlinear Dynamical Automaton (the maps ψ and Ψ are discussed in Section 4.3). Finally, a map ρ maps the Nonlinear Dynamical Automaton to a specific Recurrent Neural Network architecture with internal dynamics ζ, and identically maps encoded dotted sequences to the activation of a specific pair of neural units in the network (the maps ρ and ζ are discussed in Section 4.4).

4.1 Versatile Shifts

As we discussed in Section 3.2.1, GSs can simulate any Turing Machine (as proved by Moore, 1990, 1991), and are thus a Universal model of computation. This implies that GSs can simulate any other model of computation of power equal or inferior than that of Turing Machines. This, however, does not guarantee that the simulation will be efficient, as each machine model has its own set of atomic operations which is not necessarily easy to simulate through the operations of a different machine, requiring multiple computation steps for each step in the original machine. This is as true for abstract automata as it is in the real world: virtual machines simulating some system, for example, require more processing power to run than that necessary to run the simulated systems natively (Smith and Nair, 2005). Simulating automata through GSs will thus lead in many cases to unnecessarily complicated shift spaces, reflecting the complicated machine tables needed by the TMs themselves to simulate the original automata. We thus add more flexibility to the GS object, allowing it to simulate different atomic operations from different automata parsimoniously and in real time. We do so by relaxing some of the constraints in the model, while at the same time preserving the possibility to map its dynamics to that of NDA. We refer to this novel shift map as *Versatile Shift* (VS).

In order to present the model, we redefine the concept of *dot* in a dotted sequence. Whereas until now the dot was only used to facilitate notation (i.e. to denote the 0-th coordinate of a dotted sequence), we will now consider the dot as a meta-symbol used to concatenate two words, such that the set of dotted words can be defined as $\hat{\mathbf{A}}^* = \{v_1.v_2 \mid v_1, v_2 \in \mathbf{A}^*\}$. If we denote the set of left-infinite and right-infinite sequences respectively as $\mathbf{A}^{\mathbb{Z}^-}$ and $\mathbf{A}^{\mathbb{Z}^+}$, then a dotted sequence can be defined as a bi-infinite sequence of symbols $s \in \mathbf{A}^{\mathbb{Z}}$ where $s = w_l \; v \; w_r$ with $v \in \hat{\mathbf{A}}^*$ being a dotted word $v = v_1 \; . \; v_2$, such that $w_l v_1 \in \mathbf{A}^{\mathbb{Z}^-}$ and $v_2 w_\beta \in \mathbf{A}^{\mathbb{Z}^+}$. That is, the indices in the newly defined dotted sequence are now inherited from the dotted word v rather than specified beforehand (this may seem of little consequence at the moment, but will be important later).

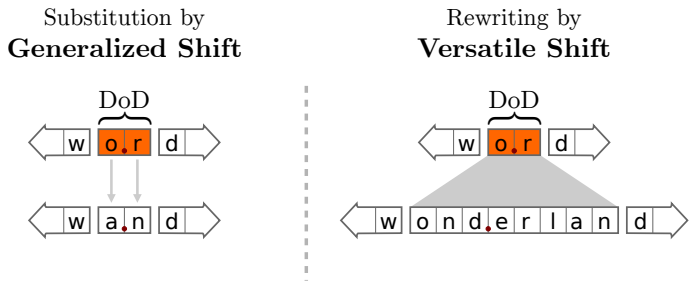


FIGURE 4.2: **Substitution by a Generalized Shift versus rewriting by a Versatile Shift.** The key difference between a Generalized Shift and a Versatile Shift is the way they perform symbol replacement on dotted sequences. Specifically, as the Figure shows, a Generalized Shift can only replace each symbol in its Domain of Dependence with a new one; a Versatile Shift can instead substitute the dotted word in its Domain of Dependence with a new dotted word, with no length restrictions. A Versatile Shift reduces to a Generalized Shift when the substituted dotted words always have the same length and indexing as the original dotted word in the Domain of Dependence.

The key difference between our novel VS and the GS is in the rewriting operation (see Figure 4.2). In fact, the GS is defined such that the joint action of a G function and of a $\oplus$ operator results in the rewriting of a finite number of symbols in the original sequence with new ones. In the VS we redefine G and $\oplus$ such that their operation results in the substitution of a dotted word of some length in the original sequence with a new dotted word of equal or different length. Through this, we add expressiveness to VSs in comparison to GSs while still maintaining the possibility to map the shift's dynamics to NDA, thus being able to simulate a greater range of models of computation in real time through piecewise affine-linear systems on the unit square. We will now introduce a formal definition of VS.

Definition 4.1. we define a Versatile Shift (VS) as a pair $M_{VS} = (\mathbf{A}^{\mathbb{Z}}, \Omega)$, with $\mathbf{A}^{\mathbb{Z}}$ being the space of dotted sequences, and $\Omega : \mathbf{A}^{\mathbb{Z}} \to \mathbf{A}^{\mathbb{Z}}$ defined by

$$\Omega(s) = \sigma^{F(s)}(s \oplus G(s)) \tag{4.1}$$

with

$$F : \mathbf{A}^{\mathbb{Z}} \to \mathbb{Z}, \quad \oplus : \mathbf{A}^{\mathbb{Z}} \times \hat{\mathbf{A}}^* \to \mathbf{A}^{\mathbb{Z}}, \quad G : \mathbf{A}^{\mathbb{Z}} \to \hat{\mathbf{A}}^*, \tag{4.2}$$

where the substitution operator "$\oplus$" substitutes the dotted word $v_1.v_2 \in \hat{\mathbf{A}}^*$ in s with a new dotted word $\hat{v_1}.\hat{v_2} \in \hat{\mathbf{A}}^*$ specified by G, while $F(s)$ determines the number and direction of the shifts. The action of F, G and $\oplus$ in a VS is determined by the contents of a finite set of consecutive cells in the original dotted sequence, which indices define the DoD of the VS.

The DoD can be specified by an open interval $(k_l, k_r) \subset \mathbb{Z}$, with $k_l \leq 0$ and $k_r \geq 0$. Additionally, we define $\mathrm{DoD}_l = (k_l, 0)$ and $\mathrm{DoD}_r = (-1, k_r)$ to be the left and right part of the complete DoD of the VS, such that $\mathrm{DoD} = \mathrm{DoD}_l \cup \mathrm{DoD}_r$. The set $V \subset \hat{\mathbf{A}}^*$ of dotted words that can appear in the DoD of a VS can be defined as $V = \left\{ v \;\middle|\; v = v_1.v_2 \in \hat{\mathbf{A}}^*,\ |v_1| = |\mathrm{DoD}_l|,\ |v_2| = |\mathrm{DoD}_r| \right\}$.

We will now present an example to illustrate the action of VSs on dotted sequences. Let us define a VS Ω_{VS} such that

$$\mathrm{DoD} = (-2, 1) = \{-1, 0\}, \quad G|_{\mathrm{DoD}} : \begin{cases} \texttt{o.r} \mapsto \texttt{a.n} \\ \texttt{a.n} \mapsto \texttt{on.derlan}, \end{cases} \quad F|_{\mathrm{DoD}} : \begin{cases} \texttt{o.r} \mapsto 0 \\ \texttt{a.n} \mapsto 1. \end{cases}$$

Then, applying Ω_{VS} to "wo.rd" two times yields

$$
\begin{aligned}
\Omega^2_{\text{VS}}(\texttt{wo.rd}) &= \Omega_{\text{VS}}\left(\sigma^{F(\texttt{wo.rd})}\left(\texttt{wo.rd} \oplus G(\texttt{wo.rd})\right)\right) \\
&= \Omega_{\text{VS}}\left(\sigma^{F(\texttt{wo.rd})}\left(\texttt{wo.rd} \oplus \texttt{a.n}\right)\right) \\
&= \Omega_{\text{VS}}\left(\sigma^{F(\texttt{wo.rd})}\left(\texttt{wa.nd}\right)\right) \\
&= \Omega_{\text{VS}}\left(\sigma^{0}\left(\texttt{wa.nd}\right)\right) \\
= \Omega_{\text{VS}}(\texttt{wa.nd}) &= \sigma^{F(\texttt{wa.nd})}\left(\texttt{wa.nd} \oplus G(\texttt{wa.nd})\right) \\
&= \sigma^{F(\texttt{wa.nd})}\left(\texttt{wa.nd} \oplus \texttt{on.derla}\right) \\
&= \sigma^{F(\texttt{wa.nd})}\left(\texttt{won.derland}\right) \\
&= \sigma^{1}\left(\texttt{won.derland}\right) \\
= \texttt{wo.nderland}
\end{aligned}
$$

where the DoD of the input string has been highlighted for clarity.

4.2 Simulation of various automata by Versatile Shifts

We will now show how Versatile Shifts can seamlessly support the simulation of a range of models of computation. Importantly, given that the mapping of non-deterministic models of computation to RNNs is outside the scope of our work, in what follows we imply determinism for all the discussed automata.

4.2.1 Finite-State Machines

It is possible to encode a FSM configuration c on a dotted sequence as

$$\Sigma_s(c) = q_t \; . \; d_{0_t} \; d_{1_t} \; \ldots \; d_{n_t} \tag{4.3}$$

where q_t, d_{0_t} and $d_{1_t} \ldots d_{n_t}$ are respectively the state, input symbol, and the rest of the unconsumed input of the FSM at time t.

Given any FSM, it is possible to construct a VS simulating it in real-time by defining a map $\Sigma_\delta : \delta_{\text{FSM}} \mapsto F, G$ between the FSM transition function δ_{FSM} and the VS F and G

functions such that

$$\begin{aligned} F|_{\mathrm{DoD}}(q_t.d_{0_t}) &= 0 \\ G|_{\mathrm{DoD}}(q_t.d_{0_t}) &= q_{t+1}.\epsilon, \end{aligned} \tag{4.4}$$

with Domain of Dependence $\mathrm{DoD} = (-2, 1) = \{-1, 0\}$ and where $q_{t+1} = \delta(q_t, d_{0_t})$.

4.2.2 Push-Down Automata

A PDA configuration c can be encoded on a dotted sequences as follows:

$$\Sigma_s(c) = \underbrace{s_{m_t} \ \ldots \ s_{0_t}}_{s_t} \ q_t \ . \ \underbrace{d_{0_t} \ \ldots \ d_{n_t}}_{d_t} \tag{4.5}$$

where q_t, d_t and s_t are respectively the state, the unconsumed input and the content of the stack of the automaton in reversed order at time t.

Given any PDA, it is possible to construct a VS simulating it in real-time by defining a map $\Sigma_\delta : \delta_{\mathrm{PDA}} \mapsto F, G$ between the FSM transition function δ_{PDA} and the VS F and G functions such that

$$\begin{aligned} F|_{\mathrm{DoD}}(s_{0_t} q_t.\kappa) &= 0 \\ G|_{\mathrm{DoD}}(s_{0_t} q_t.\kappa) &= \begin{cases} \epsilon \ q_{t+1} \ . \ \epsilon & \text{if } \chi = \epsilon \\ s_{0_t} \ \chi \ q_{t+1} \ . \ \epsilon & \text{otherwise.} \end{cases} \end{aligned} \tag{4.6}$$

with Domain of Dependence $\mathrm{DoD} = (-3, 1) = \{-2, -1, 0\}$, and where $(q_{t+1}, \chi) = \delta(q_t, \kappa, s_{0_t})$,

4.2.3 Top-Down Recognizers

In this work we use Top-Down Recognizers to process locally unambiguous non-left-recursive CFGs [1]. This class of Top-Down Recognizers only have one state q_0, and we can thus describe their machine configuration without referring to the current state. It is possible to encode a (single-state) TDR configuration c on a dotted sequences as follows:

$$\Sigma_s(c) = \underbrace{s_{m_t} \ \dots \ s_{0_t}}_{s_t} \ . \ \underbrace{d_{0_t} \ \dots \ d_{n_t}}_{d_t} \tag{4.8}$$

where d_t and s_t are respectively the unconsumed input and the content of the stack of the automaton in reverse order at time t. Similarly, simpler VSs than those needed to simulate PDAs can be constructed from a TDR's transition function, by defining the map $\Sigma_\delta : \delta_{\text{TDR}} \mapsto F, G$ such that

$$\begin{aligned} F|_{\text{DoD}}(s_{0_t}.\kappa) &= 0 \\ G|_{\text{DoD}}(s_{0_t}.\kappa) &= \chi.\epsilon \end{aligned} \tag{4.9}$$

with Domain of Dependence $\text{DoD} = (-2, 1) = \{-1, 0\}$, and where $(q_0, \chi) = \delta(q_0, \kappa, s_{0_t})$.

4.2.4 Turing Machines

It is possible to encode a TM configuration c on a dotted sequence as shown in 3.6, i.e as

$$\Sigma_s(c) = \underbrace{\dots \ d_{-2_t} \ d_{-1_t}}_{l_t} \ q_t \ . \ \underbrace{d_{0_t} \ d_{1_t} \ d_{2_t} \ \dots}_{r_t},$$

where l_t describes the part of the tape to the left of the read-write head, r_t describes the part to its right, q_t describes the current state of the machine controller, and the central dot denotes the current position of the read-write head, i.e. d_{0_t}, the symbol to its right.

[1] A recursive CFG is defined as a CFG which includes at least one rule of the form $A \to uAv$, which expands a non-terminal symbol A into a string that contains the same non-terminal; a left-recursive CFG is a recursive CFG where all such rules are in the form $A \to Aw$; A CFG is locally unambiguous if no two rules expand the same nonterminal.

Note that, in relation to the CFG grammar $G_{\text{CF}} = (\mathbf{N}, \mathbf{T}, R, \mathtt{S})$ from which the TDR is derived, the previous equation implies that

$$G|_{\text{DoD}} : \begin{cases} a \, .a & \mapsto & \epsilon \, .\epsilon \\ X.a & \mapsto & w.\epsilon \end{cases} \tag{4.7}$$

for all $a \in \mathbf{T}$, $(X \to w) \in \mathbf{R}$.

A VS simulating a TM in real-time can be constructed from the TM's transition function by defining the Domain of Dependence to be DoD $= (-3, 1) = \{-2, -1, 0\}$, and G and F so that, given $\delta : (q_t, d_{0_t}) \mapsto (q_{t+1}, \hat{d}_{0_t}, m)$,

$$\begin{aligned} G: & \begin{cases} d_{-1_t}\ q_t\ .\ d_{0_t} & \mapsto & d_{-1_t}\ \hat{d}_{0_t}\ .\ q_{t+1} & \text{if } m = \mathcal{R} \\ d_{-1_t}\ q_t\ .\ d_{0_t} & \mapsto & q_{t+1}\ d_{-1_t}\ .\ \hat{d}_{0_t} & \text{if } m = \mathcal{L} \end{cases} \\ F: & \begin{cases} d_{-1_t}\ q_t\ .\ d_{0_t} & \mapsto & -1 & \text{if } m = \mathcal{R} \\ d_{-1_t}\ q_t\ .\ d_{0_t} & \mapsto & +1 & \text{if } m = \mathcal{L} \end{cases} \end{aligned} \tag{4.10}$$

for all $d_{-1_t} \in \mathbf{N}$.

4.3 Dynamical Systems from Versatile Shifts

We will now show that the action of a VS can be mapped to that of a piecewise affine-linear system on the symbologram, yielding a NDA. This is achieved through a map Ψ that, given the VS and two Gödelizations (ψ_x, ψ_y) mapping its dotted sequences to points on the symbologram, derives the rectangular partition $P = \{D^{i,j}\}$ of the unit square and the set of linear transformations $\Phi^{i,j}$ defining the NDA.

Specifically, the P partition is defined by the Gödelization of cylinder sets containing sequences which agree on their DoD symbols. The Gödelization of each cylinder set is a rectangle $D^{i,j} = I_i \times J_j$, where $I_i = [\xi_i, \xi_{i+1}), J_j = [\eta_j, \eta_{j+1})$. The I_i and J_j intervals are defined respectively by the Gödelization of the left and right constituents of dotted sequences in the cylinder set. The lower bound of each I_i interval (equivalently, the upper bound of each I_{i-1} interval) can be derived as the minimum of the Gödelization of the cylinder set for a given content of the DoD. In fact, if we let $z_l = |\text{DoD}_l|$ be the number of symbols in the left DoD of the VS and $\gamma_x(r^*) = \min_r \big(\gamma_x(r)\big)$ be the minimum of the γ_x enumerating function for the Gödelization of the left constituents, we can derive the

minimum of the Gödelized left constituent as

$$\begin{aligned}
\xi_i &= \min_{w} \left(\psi_x(w_l^{\mathrm{R}})\right) \\
&= \sum_{k=1}^{z_l} \gamma_x(r_k) g^{-k} + \sum_{k=z_l+1}^{\infty} \gamma_x(r^*) g^{-k} \\
&= \sum_{k=1}^{z_l} \gamma_x(r_k) g^{-k} + \gamma_x(r^*) \left(\frac{g}{1-g} - \sum_{k=1}^{z_l} g^{-k} \right) \\
&= \sum_{k=1}^{z_l} \gamma_x(r_k) g^{-k} + \gamma_x(r^*) \left(\frac{g}{1-g} - \frac{g - g^{z_l+1}}{1-g} \right) \\
&= \sum_{k=1}^{z_l} \gamma_x(r_k) g^{-k} + \gamma_x(r^*) \frac{g^{z_l+1}}{1-g}
\end{aligned} \tag{4.11}$$

where we have used that, given $g^{-1} < 1$, $\sum_{k=1}^{\infty} g^{-k}$ is a geometric series converging to $\frac{g}{1-g}$, and that, for $a, b \in \mathbb{N}$, $\sum_{k=a}^{b} g^k = \frac{g^a - g^{b+1}}{1-g}$. The same argument as in Equation 4.11 can be used to show that $\eta_j = \sum_{k=1}^{|\mathrm{DoD}_r|} \gamma_y(r_k) g_y^{-k} + \gamma_y(r^*) \frac{g_y^{|\mathrm{DoD}_r|+1}}{1-g_y}$. With $\xi_{m+1} = \eta_{n+1} = 1$ as boundary conditions, the Gödelized cylinder sets are disjoint intervals covering all of the unit square, thus defining the partition P of the NDA.

Each of the D_{ij} Gödelized cylinder sets is associated with a specific dotted word in the DoD of the VS, and thus to a specific substitution and shift by the VS. We will now show that each DoD-specific substitution and shift by a VS can be represented as a linear transformation on the symbologram, enabling us to obtain the $\Phi^{i,j}$ set of linear transformations of the NDA.

We have previously proved that rewriting and shift operations on dotted sequences by a GS can be mapped to push and pop operations on their one-sided infinite constituents, and that push and pop operations can be represented on the symbologram as linear transformations on the Gödelized dotted sequences. By introducing the VS, we extended the GS rewriting operation with a more general substitution operation, which can substitute a dotted word in the original dotted sequence with any other arbitrary dotted word of similar or different length. We will now show that this rewriting operation can also be mapped to push and pop operations on the one-sided infinite constituents of the original dotted sequence. This

result implies that substitutions and shifts by a VS can be represented as affine-linear transformations on the symbologram.

Let $s \oplus G(s) = w_l u.vw_r \oplus \hat{u}.\hat{v}$ be a substitution replacing the dotted word $u.v$ in s with the dotted word $\hat{u}.\hat{v}$, then $s \oplus G(s)$ can be straightforwardly mapped to pop and push operations on $(w_l u)^{\mathrm{R}}$ and vw_r, the one-sided constituents of the original dotted sequence s, as follows:

$$\begin{aligned} w_l u.vw_r \oplus \hat{u}.\hat{v} &= \left(\left(\ominus^{|u|} u^{\mathrm{R}} {w_\alpha}^{\mathrm{R}}\right) \odot \hat{u}^{\mathrm{R}}\right)^{\mathrm{R}} . \left(\left(\ominus^{|v|} vw_r\right) \odot \hat{v}\right) \\ &= \left(w_l^{\mathrm{R}} \odot \hat{u}^{\mathrm{R}}\right)^{\mathrm{R}} . \left(w_r \odot \hat{v}\right) \\ &= w_l \hat{u}.\hat{v} w_r . \end{aligned}$$

We are thus able to represent a VS on the symbologram by deriving the parameters of the affine-linear transformations associated with its operations. In fact, given that i) a substitution and shift by VS can be represented as the composition of push and pop operations, ii) push and pop operations can be represented on the symbologram as affine-linear transformations (as shown in Section 3.3.2), and iii) the composition of affine-linear transformations is an affine-linear transformation, then each VS substitution and shift can be represented as a bi-dimensional affine-linear transformation on the symbologram.

4.3.1 A refined Gödelization

In the coming Sections, we will often encounter dotted sequences $w_l.w_r$ representing machine configurations such that the index -1 only ever contains a state symbol, with the rest of the dotted sequence only containing tape symbols (as discussed in Sections 4.2.1, 4.2.2 and 4.2.4). In order to cover all of the available representational space $[0, 1]^2$, we can define a refined Gödelization as

$$\psi_x(s) := \gamma_q(d_1) n_q^{-1} + \sum_{k=1}^{\infty} \gamma_s(d_{k+1}) n_s^{-k} n_q^{-1}, \tag{4.12}$$

with γ_q and γ_s respectively enumerating the set of states Q and that of tape symbols $\mathbf{A}$, and where $n_q = |Q|$, $n_s = |\mathbf{A}|$. It is trivial to show, through the same arguments presented in Sections 3.3.1, 3.3.2 and 4.3, that the refined Gödelization of a VS preserving

the machine configuration encoding in the original dotted sequence (i.e. with the resulting sequence only containing state symbols at index -1 and never elsewhere) is akin to linear transformations on the original refined Gödelization of the dotted sequence.

4.4 NDA to Recurrent Neural Networks

We have introduced VSs on dotted sequences and shown that their symbolic dynamics can simulate a range of models of computation in real-time, and can be mapped to point dynamics of NDA, discrete-time piecewise affine-linear dynamical systems on the unit square. In this Section, we will present a mapping between NDA and Recurrent Neural Networks (RNNs). This allows us to simulate VSs in real time, and thus a variety of models of computation in real-time, through Neural Network dynamics.

The mapping defines a RNN architecture which is modular and transparently reproduces the structure of NDA through three layers, the machine configuration layer (MCL), the branch selection layer (BSL) and the linear transformation layer (LTL), as shown in Figure 4.4. Specifically, the action of the three layers can be summarized as follows:

Machine Configuration Layer. The MCL encodes the state of the NDA, and thus the symbolic data of the simulated machine, and functions as a read-out layer in this architecture. Given that the state space X of the NDA is the unit square, the MCL only needs two neural units to maintain the machine configuration encoding, units which we will refer to as c_x and c_y. The MCL has a forward connection to the BSL and the LTL.

Branch Selection Layer. The BSL receives as an input the activation values of the c_x and c_y MCL units, and outputs to the LTL. The BSL is divided in two sets of units, b_x and b_y, each connected to one of the two MCL units, and acting as a switching system with regards to the LTL. In relation to the NDA dynamics, at each time step the BSL determines to which $D^{i,j}$ cell the machine configuration encoded in the MCL activation belongs. As a result, thanks to an interplay between inhibitory and

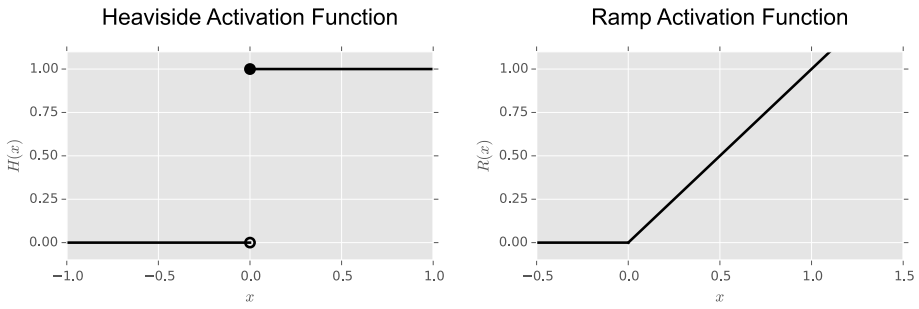


FIGURE 4.3: **Activation functions for units in the proposed architecture.** The Ramp activation function (on the right) is used by units in the Machine Configuration Layer and the Linear Transformation Layer of the network. The Heaviside activation function (on the left) is instead used by units in the Branch Selection Layer.

excitatory connections to the LTL, only two units in the LTL are allowed to stay active and perform their function.

Linear Transformation Layer. The LTL receives input both from the MCL and the BSL, and has output recurrent connections to the MCL. The LTL can be divided in pairs of t_x and t_y neural units, each applying one of the $\Phi^{i,j}$ decoupled bi-dimensional affine-linear transformations. Each t_x neural unit is connected to the c_x MCL unit, and each t_y unit to the c_y MCL unit. In this way, at each time step the BSL makes sure only a pair of LTL units is active, and the pair applies its $\Phi^{i,j}(x, y)$ affine-linear transformation to the (c_x, c_y) input from the MCL, reproducing in vectorial space a symbolic operation by the original simulated machine, and updating the MCL activation with the new encoded machine configuration.

The activation functions for the neural units in this architecture are the Heaviside (H) and the Ramp (R) activation functions, defined as follows (and shown in Figure 4.3):

$$H(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \qquad R(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases} \tag{4.13}$$

To formally characterize the construction of our architecture, we define a map ρ between NDA and RNNs, with the objective of mapping the NDA dynamics Φ to neural dynamics ζ, as

$$\zeta = \rho(\mathcal{I}, \mathcal{A}, \Phi, \Theta), \tag{4.14}$$

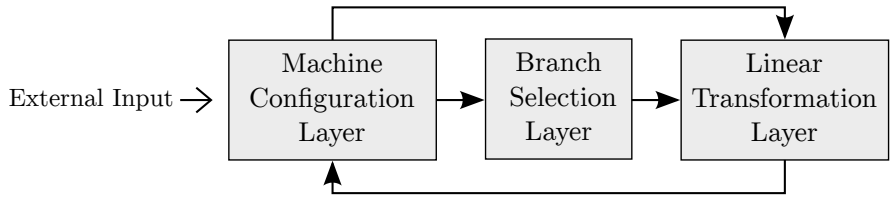


FIGURE 4.4: **Layer connectivity in the Recurrent Neural Network architecture.** In the network, the Machine Configuration Layer sends output to the Branch Selection Layer and the Linear Transformation Layer; the Branch Selection Layer receives input from the Machine Configuration Layer and outputs to the Linear Transformation Layer; the Linear Transformation Layer receives input from the Machine Configuration Layer and the Branch Selection Layer, and has a recurrent output connection to the Machine Configuration Layer. In this architecture, the Machine Configuration Layer acts as a read-out layer (as it stores the encoded machine configuration of the simulated automaton), and is initialized at the beginning of the computation through external input.

where $\mathcal{I}_{2\times 2} = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$ is an identity matrix identically mapping the NDA initial conditions to the MCL initial conditions, $\mathcal{A}$ is the weight matrix of the network architecture, and Φ and Θ are respectively the flow and switching rule of the NDA to be simulated. Furthermore, ζ can be divided in three distinct components $\zeta = (\zeta_{\text{MCL}}, \zeta_{\text{BSL}}, \zeta_{\text{LTL}})$, each corresponding to the dynamics of one of the layers in the network.

In the following Sections, we will discuss the role for the ρ map for each of the three layers in the network.

4.4.1 Machine Configuration Layer

The MCL encodes at each time step the state of the simulated NDA, which in turn encodes the Gödelized representation of the symbolic data of the machine that the NDA simulates. Given that the Gödelized representation of machine configurations is a point on the symbologram (as discussed in Section 3.3.1), i.e. the unit square, the MCL only requires two neural units, c_x and c_y, to store the encodings. In addition to storing the encoded symbolic data of the simulated machine, and thus acting as a read-out layer for the RNN, the MCL also makes this information available to the rest of the network through forward connections to the BSL and LTL (see Figure 4.5 for the detailed connectivity).

The MCL is initialized at the beginning of the computation with the encoded initial configuration of the simulated machine. Specifically, the NDA initial conditions $\big(\psi_x(w_l^{\text{R}}), \psi_y(w_r)\big)$

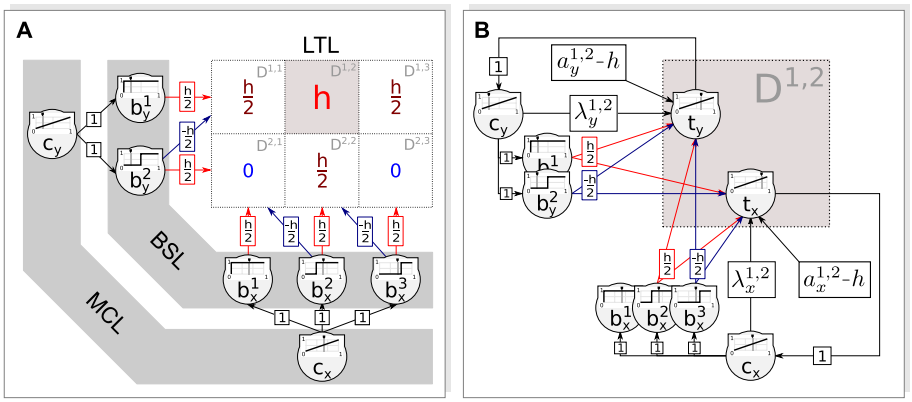


FIGURE 4.5: **Detailed connectivity of network architecture.** The Figure shows the connectivity of an example network simulating a NDA with 3×2 cells $D^{i,j}$. In panel (A), the MCL c_x and c_y units have a forward connection with weight 1 to respectively the $b_x = \{b_x^1, b_x^2, b_x^3\}$ and $b_y = \{b_y^1, b_y^2\}$ units. The BSL units are instead connected with a specific excitatory-inhibitory pattern to the relevant pairs of units in the LTL, such that only a given pair (in this case the pair corresponding to cell $D^{1,2}$ in the simulated NDA) receives a cumulative input of h from the BSL, with all other pairs receiving either $\frac{h}{2}$ or 0. As shown in panel (B), each pair has an intrinsic inhibitory bias component of h, such that only pair the $D^{1,2}$ pair in this example receives enough input from the BSL to fully counteract the inhibition. Additionally, each $t_x^{i,j}$ unit in the LTL is connected with weight $\lambda_x^{i,j}$ to the c_x unit in the MCL, and has an intrinsic bias component of $a_x^{i,j}$; each $t_y^{i,j}$ unit in the LTL is instead connected with weight $\lambda_y^{i,j}$ to the c_y unit and has an intrinsic bias component of $a_y^{i,j}$. This allows each pair, when de-inhibited by the BSL, to apply the corresponding $\Phi^{i,j} = (\lambda_x^{i,j} x + a_x^{i,j}, \lambda_y^{i,j} x + a_y^{i,j})$ linear transformation of the simulated NDA.

are directly encoded as the MCL initial activation values by the ρ map as

$$(c_x, c_y) = \big(\psi_x(w_l^{\mathrm{R}}), \psi_x(w_r)\big) \equiv \zeta_{\mathrm{MCL}}(0) = \rho(\mathcal{I}, \cdot, \cdot, \cdot)\big|_{\left(\psi_x(w_l^{\mathrm{R}}), \psi_x(w_r)\right)} \tag{4.15}$$

where, to avoid abusing notation, we denote the activation of the c_x and c_y units with the same symbols as the units themselves. For each computation step following the initialization (i.e. for each $t > 0$), the MCL units are activated by input from the t_x and t_y LTL units through a R ramp activation function, such that $\zeta_{\mathrm{MCL}} \equiv (c_x, c_y) = \left(R\left(\sum_i t_x^i\right), R\left(\sum_j t_y^j\right)\right)$.

4.4.2 Branch Selection Layer

The BSL implements the $\Theta(x, y)$ switching rule of the simulated NDA, controlling which of the affine-linear transformations $\Phi^{i,j}$ is applied at each time step by the LTL by coordinating the dynamic switching of LTL units. That is, the BSL uses an interplay between

excitatory and inhibitory connections to make sure that, given an activation of (c_x, c_y) in the MCL, only the pair of $(t_x^{i,j}, t_y^{i,j})$ LTL units associated with the application of the $\Phi^{i,j}$ affine-linear transformation is activated, as prescribed by the NDA switching rule $\Theta(c_x, c_y) = (i, j)$. The switching rule is thus mapped by $\rho(\cdot)$ to the dynamics of this neural layer as follows:

$$\zeta_{\text{BSL}}(x, y) = \rho(\cdot, \cdot, \cdot, \Theta(x, y) = \{i, j\}). \tag{4.16}$$

The NDA switching rule Θ maps a point on the unit square to the indices of the cell in the NDA P partition which the point belongs to, i.e. if $(x, y) \in D^{i,j}$ then $\Theta(x, y) = (i, j)$. Note that, as $D^{i,j} = I_i \times J_j$ is a rectangular cell, then $\Theta(x, y) = (i, j)$ when $x \in I_i$ and $y \in J_j$. The BSL performs these two checks through two separate sets of Heaviside neural units, where a b_x set of m units checks for which i $x \in I_i$, and a b_y set of n units checks for which j $y \in J_j$. Each b_x^i unit receives a synaptic projection from the MCL c_x unit and has bias $-\xi^i$, whereas each b_y^j unit receives a projection from the c_y unit and has bias $-\eta_j$, such that

$$\begin{aligned} b_x^i &= H(c_x - \xi^i) \qquad \text{with} \qquad \xi^i = \min(I_i), \\ b_y^j &= H(c_y - \eta^j) \qquad \text{with} \qquad \eta^j = \min(J_j). \end{aligned} \tag{4.17}$$

where b_x^i and b_y^j denote the activation of the b_x^i and b_y^j units. The biases $-\xi_i$ and $-\eta_j$ function as an activation threshold, where the biased Heaviside neural units are active with activation 1 only if their input is equal to or exceeds the threshold (see Equation 4.13). The effect is that, given an input $(c_x, c_y) \in D^{k,z}$ from the MCL, all units b_x^i and b_y^j in the BSL with $i \leq k$ and $j \leq z$ would be triggered active.[2]

To attain the capability of selectively activating pairs (t_x^i, t_y^j) of LTL units, the BSL is connected to the LTL with a specific pattern of excitatory and (lateral) inhibitory connections, which interaction with a strong inhibitory bias in the LTL determines the pair to be activated. That is, each unit in the LTL is naturally inactive through the use of a strong negative bias h, which the BSL fully counteracts only for the specific pair of "selected" units.

[2] As a side note, this means that the sum of the activations in the BSL b_x and b_y groups $(\sum_i b_x^i, \sum_j b_y^j) = (i, j)$, i.e. the two groups transparently encode $\Theta(x, y) = (i, j)$.

Each neural unit b_x^k establishes an excitatory connection with weight $\frac{h}{2}$ with all $(t_x^{k,j}, t_y^{k,j})$ units in the LTL for $1 \geq j \geq n$, i.e. the units corresponding to the linear transformations $\Phi^{k,j}$ associated with cells $D^{k,j}$ in the simulated NDA. Moreover, each b_x^k unit also establishes an inhibitory connection with weight $\frac{h}{2}$ with all $(t_x^{k,j-1}, t_y^{k,j-1})$ units in the LTL for $2 \geq j \geq n$ (implementing a kind of lateral inhibition). Specularly, each unit b_y^z is connected with weight $\frac{h}{2}$ to all $(t_x^{i,z}, t_y^{i,z})$ LTL units, and with weight $-\frac{h}{2}$ to all $(t_x^{i-1,z}, t_y^{i-1,z})$ LTL units with $2 \geq i \geq m$, i.e. the units corresponding respectively to the linear transformations $\Phi^{i,z}$ and $\Phi^{i-1,z}$ associated with cells $D^{i,z}$ and $D^{i-1,z}$ in the simulated NDA.

The two b_x and b_y groups act together to fully counterbalance the strong LTL h bias for a pair of LTL units. Specifically, it is useful to think about the LTL as a bi-dimensional grid of pairs of units (each pair corresponding to a cell $D^{i,j}$ of the P unit square partition of the simulated NDA). At each step of the computation, the b_x group then counterbalances half of the h inhibition for a row k of LTL units, whereas the b_y counterbalances half of the h inhibition for a column z of the LTL. The pair of units at the crossing between the k row and the z column is then fully de-inhibited by the BSL. More formally, each pair of LTL units $(t_x^{i,j}, t_y^{i,j})$ receives input from the BSL as

$$\begin{aligned} B_x^i &= b_x^i \tfrac{h}{2} + b_x^{i+1} \tfrac{-h}{2} \\ B_y^j &= b_y^j \tfrac{h}{2} + b_y^{j+1} \tfrac{-h}{2}, \end{aligned} \tag{4.18}$$

where the input sum

$$B_x^i + B_y^j = \begin{cases} h & \text{if } (c_x, c_y) \in D_{i,j} \\ \frac{h}{2} & \text{if } c_x \in I_i, c_y \notin J_j \quad \text{or} \quad c_x \notin I_i, c_y \in J_j \\ 0 & \text{if } (c_x, c_y) \notin D_{i,j} \end{cases} \tag{4.19}$$

only fully de-inhibits the a LTL unit if it reaches the value h. In other words if $(c_x, c_y) \in D^{k,z}$, then $B_x^k + B_y^z = h$ only for the LTL pair $(t_x^{k,z}, t_y^{k,z})$. For all other pairs, $B_x^k + B_y^z$ does not reach h, but is either equal to $\frac{h}{2}$ or 0. A pictorial representation of this mechanism is shown in Figure 4.5.

4.4.3 Linear Transformation Layer

The LTL is composed of pairs of units, each one implementing one of the $\Phi^{i,j}(x,y) = (\lambda_x^{i,j}x + a_x^{i,j},\ \lambda_y^{i,j}y + a_y^{i,j})$ bi-dimensional affine-linear transformations of the simulated NDA, where each unit in the pair applies one component of the decoupled transformation. That is,

$$(t_x^{i,j}, t_y^{i,j}) \quad = \quad \zeta_{\text{LTL}}^{i,j}(x,y) = \rho(\cdot,\cdot,\Phi^{i,j}(x,y),\cdot). \tag{4.20}$$

From the point of view of the symbolic computation the RNN simulates, each pair essentially implements the application of a specific symbolic operation on input data (one cell of the machine transition table), here encoded through input from the MCL to the pair. Through the transformation mediated by the pair (if the pair is selected by the BSL), an encoded output string is obtained, which is then fed to the MCL for the next time step as the updated machine configuration. The affine-linear transformation is applied through a combination of synaptic computation and intrinsic dynamics of the units in the pair, only triggered when enough excitatory input is received from the BSL. In fact, for each pair of units,

$$\begin{aligned} t_x^{i,j} &= R(\lambda_x^{i,j}c_x + a_x^{i,j} - h + B_x^i + B_y^j) \\ t_y^{i,j} &= R(\lambda_y^{i,j}c_y + a_y^{i,j} - h + B_x^i + B_y^j), \end{aligned} \tag{4.21}$$

where R is the ramp function defined in Equation 4.13, and where h is defined such that

$$-\frac{h}{2} \leq -\max_{i,j,k}(a_k^{i,j} + \lambda_k^{i,j}). \tag{4.22}$$

As summarized in Equation 4.21, the activation of a pair $(t_x^{i,j}, t_y^{i,j})$ is only equal or greater than 0 when $B_x^i + B_y^j = h$ (i.e. when $B_x^i = B_y^j = \frac{h}{2}$, as discussed in Section 4.4.2). In case the BSL selects the pair and thus fully counterbalances the natural inhibition of the units, the activation of the pair in Equation 4.21 can be rewritten simply as $(t_x^{i,j}, t_y^{i,j}) = \left(R\left(\lambda_x^{i,j}c_x + a_x^{i,j}\right), R\left(\lambda_y^{i,j}c_y + a_y^{i,j}\right)\right)$, where the input from the MCL (c_x, c_y) is modulated synaptically by weights $(\lambda_x^{i,j}, \lambda_y^{i,j})$, and an intrinsic constant neural dynamics $(a_x^{i,j}, a_y^{i,j})$ is added in the form of bias, completing the bi-dimensional affine-linear transformation.

4.4.4 Number of units in the architecture

Let us consider a network simulating a NDA with a partition P of the unit square composed by cells $D^{i,j} = I_i \times J_j$, with $1 \leq i \leq m$ and $1 \leq j \leq n$, for a total of $m \times n$ cells. Given the way our architecture is constructed, we can easily derive the number of units in a the network. In fact, for each layer:

MCL. the MCL only comprises of 2 units, c_x and c_y, each encoding one of the two one-sided constituents of a configuration dotted sequence, as discussed in Section 4.4.1.

BSL. the BSL contains two groups of units, b_x and b_y, where the number of units in b_x is equal to m, i.e. the number of I_i intervals of the NDA, and the number of units in b_y is equal to n, i.e. the number of J_j intervals of the NDA, as discussed in Section 4.4.2.

LTL. the LTL contains a pair (t_x, t_y) of units for each cell $D^{i,j}$ in the NDA P partition of the unit square, as discussed in 4.4.3. The total number of units in the LTL is thus equal to $2 \cdot m \cdot n$.

The total number of units in a network simulating a given NDA is thus equal to

$$n_{\text{units}} = \underbrace{2}_{\text{MCL}} + \underbrace{n+m}_{\text{BSL}} + \underbrace{2 \cdot n \cdot m}_{\text{LTL}}. \tag{4.23}$$

4.5 Examples

We will now present three examples of the mapping of automata computation to RNN dynamics. In the first example, we show that through Versatile Shifts and NDA, we are able to simulate the computation performed by Turing Machines in real-time through neural computation (thus implying that the architecture we present is indeed universal). In the second example, we will present the implementation of a Central Pattern Generator as a Finite State Machine, and show that the RNN resulting from the mapping of the Finite State Machine transparently produces the desired pattern in the form of spatial-temporal

Symbols	States	
	q_{even}	q_{odd}
1	$(q_{odd}, 1, \mathcal{R})$	$(q_{even}, 1, \mathcal{R})$
$\sqcup$	$(q_{acc}, \sqcup, \mathcal{L})$	$(q_{rej}, \sqcup, \mathcal{L})$

TABLE 4.1: **Transition table for Turing Machine recognizing the language of unary strings composed by an even number of ones.**

localized patterns of activation; we will also discuss the implications of these results for the field of robotic locomotion. In the third example, we construct a network of interactive automata performing the parsing of ambiguous sentences, and show that the modular construction of the network architecture we define easily accommodates the mapping of the network of automata to a neural network simulating it in real time; additionally, we show that by doing so, we make it possible to extract observables that can be put in comparison with electrophysiological data from linguistics experiments.

4.5.1 Odd vs even strings recognizer

To summarize the methods presented in this Chapter, we present a simple example on TMs, with a machine recognizing unary strings containing an even number of 1's (and, conversely, rejecting unary strings with an odd number of 1's). We construct the machine to have four states. Two states, q_{even} and q_{odd}, are used to keep track of whether the number of 1's seen until that moment is even or odd, with the machine switching state for each new 1 it reads. Depending on which between the q_{even} and q_{odd} states the machine is in once the reading of the whole input string is completed, the machine will go to a q_{acc} accept state or a q_{rej} reject state. As the strings are unary, the set of tape symbols is defined to be simply $\mathbf{N} = \{\sqcup, 1\}$, whereas the δ transition function is shown in Table 4.1. We construct a VS simulating this machine as outlined in Section 4.2.4. Given that the left constituents of the dotted sequences encoding machine configuration always contain a state as first symbol and always tape symbols in all other positions, we can use a refined

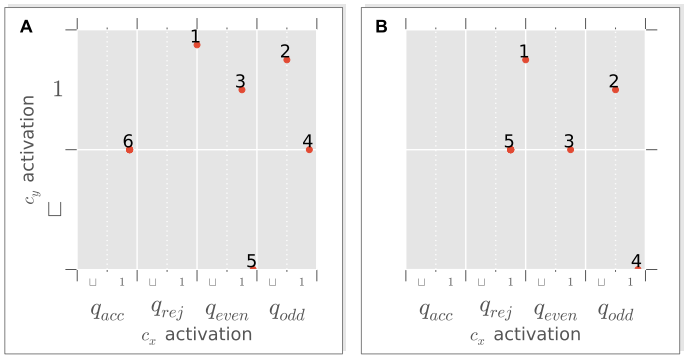


FIGURE 4.6: **Activation of c_x and c_y units in the Machine Configuration Layer as points on the symbologram**, for an input string of even ones `1111` (A) and odd ones `111` (B). Note how the point dynamics alternates between cells corresponding to the q_{even} and q_{odd} symbols in the DoD. (A) The network is initialized with an encoded input string of even ones, and thus ends its computation with a MCL activation corresponding to the encoded $\texttt{111}q_{\text{acc}}\texttt{.1}$ machine configuration. (B) The network is initialized with an encoded string of odd ones, and thus ends its computation on the encoded $\texttt{11}q_{\text{rej}}\texttt{.1}$ machine configuration.

Gödelization as discussed in Section 4.3.1, yielding the following encodings

$$\psi_x(s) := \gamma_q(d_1)n_q^{-1} + \sum_{k=1}^{\infty} \gamma_s(d_{k+1})n_s^{-k}n_q^{-1}, \qquad \psi_y(s) := \sum_{k=1}^{\infty} \gamma_s(d_k)n_s^{-k},$$

where d_i is the i-th symbol in the one-sided infinite component to be Gödelized, γ_q and γ_s are enumerating functions for respectively states and tape symbols, and $n_q = |Q|$ and $n_s = |\mathbf{N}|$ are the number of states and tape symbols of the machine, respectively. In particular, we define the γ_q, γ_s enumerating functions as

$$\gamma_q(x) = \begin{cases} 0 & \text{if } x = \sqcup, \\ 1 & \text{if } x = \texttt{1}, \end{cases} \qquad \gamma_s(x) = \begin{cases} 0 & \text{if } x = q_{\text{acc}}, \\ 1 & \text{if } x = q_{\text{rej}}, \\ 2 & \text{if } x = q_{\text{even}}, \\ 3 & \text{if } x = q_{\text{odd}}. \end{cases}$$

As discussed in Section 4.3, the VS can be mapped to a NDA on the symbologram through the Gödelization of its symbolic space, the partition of the symbologram in cells $D^{i,j}$ induced by the VS DoD, and the set of bi-dimensional affine-linear transformations defined on each cell by the Gödelized substitution and shift of the VS.

Through the methods presented in Section 4.4 we are then able to map the obtained NDA to a RNN, and thus to simulate the TM in real time via the RNN dynamics. The

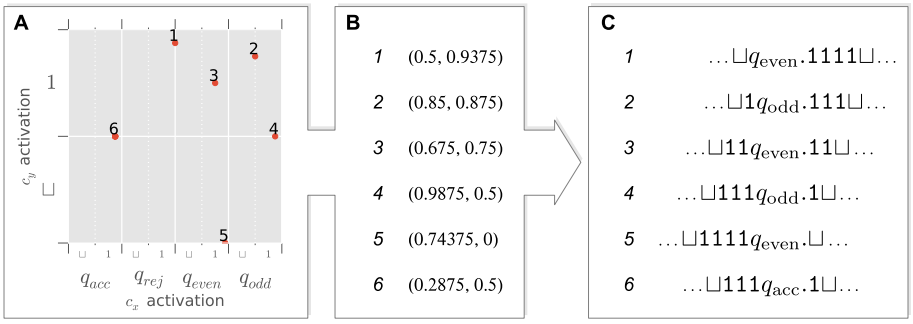


FIGURE 4.7: **Decoding of point dynamics on the symbolgram.** (A) The MCL activation dynamics on the symbolgram for an input string of even ones 1111 corresponds to (B) pairs of numbers in $[0, 1]^2 \subset \mathbb{R}^2$ which store the Gödelization of a dotted sequence representing a machine configuration. The Gödelized configuration can thus be decoded for each step of the computation, (C) retrieving the symbolic dynamics of the simulated machine.

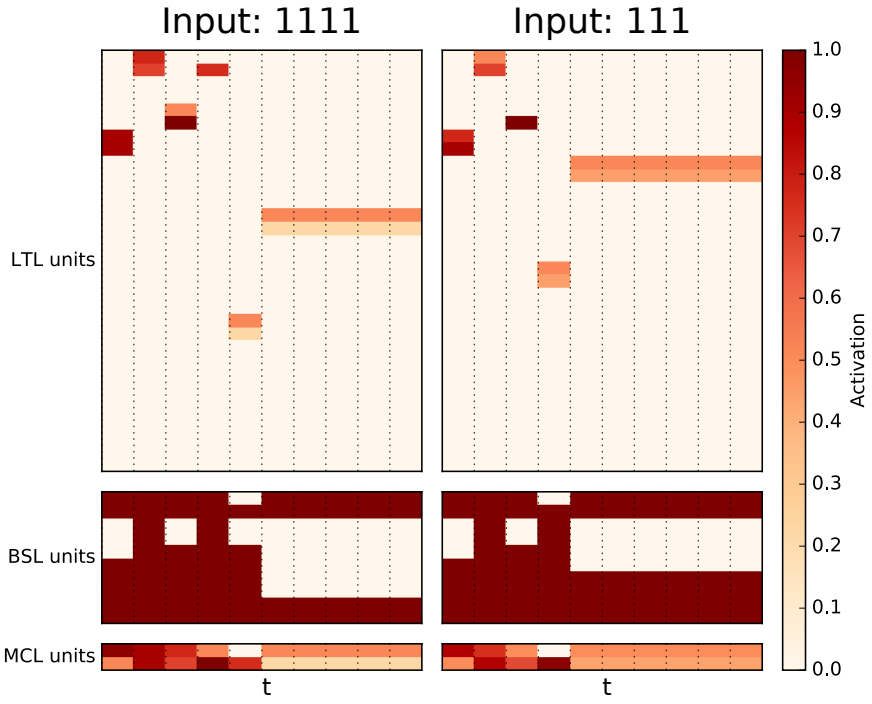


FIGURE 4.8: **Full network activation for even and odd encoded input sequence**. The Figure shows the activation in time of each of the 44 units in the network, for the two input sequences tested in the example.

resulting network is composed by 44 neural units (see Equation 4.23), and its dynamics can be observed in Figure 4.6 (only the MCL activation is reported), and Figure 4.8 (all activations reported). By decoding the MCL activation at each time step, we can fully recover the symbolic dynamics of the simulated machine, as shown in Figure 4.7. Note that in this example, we enforce a fixed point halting condition; that is, we consider the RNN computation complete when its dynamics reaches a fixed point, where it stays forever.

4.5.2 Central Pattern Generator

Central Pattern Generators (CPGs), i.e. neur(on)al networks producing rhythmic patterns in the absence of rhythmic input, offer many advantages with regards to the control of robotic locomotion, as argued in Ijspeert (2008). In his paper, the author identifies several advantages of CPGs as locomotion controllers, in fact CPGs

1. produce rhythmic patterns which are robust to perturbation of the system's state variables,
2. are suitable for distributed implementations (as in modular robotics),
3. allow for the meaningful control of the pattern they produce through a few high-level parameters,
4. can integrate sensory feedback through coupling terms in the differential equations,
5. often work well with learning and optimization algorithms.

The key issue of using CPGs as locomotion controllers in robots is that, as underlined by Ijspeert, designing a CPG to produce a given pattern is not an easy task; a sound general design methodology is still lacking, as well as a strong theoretical grounding for the description of CPGs in the general case.[3]

On the other hand, FSMs are a widespread approach in the construction of controllers for robotic locomotion, as they are trivial to describe, design, implement, and debug (see Alvarez-Alvarez et al., 2012, Collins and Ruina, 2005 for examples of FSMs as locomotion controllers in articulated robots). Additionally, they are very well understood, and their relation with animal locomotion has long been characterized (McGhee, 1968).

In this example, we outline a methodology for constructing arbitrary discrete-time CPGs starting from given patterns. This is done by first designing a FSM producing the desired pattern, representing it as a VS as shown in Section 4.2.1, mapping the VS to a NDA through Gödelization as shown in Section 4.3, and finally the NDA to a RNN through the

[3] Note that the theoretical concerns advanced here are not relative to the biological plausibility of CPGs, but to the construction of CPGs to produce arbitrary desired patterns, and how, in the general case, key pattern parameters can be controlled by manipulation of the system's parameters.

methods described in 4.4. Because of the modularity of the RNN architecture we introduce, it is possible to observe the production of the desired patterns as peaks in activation of localized assemblies in the network.

In order to demonstrate the methods, we qualitatively model the results of an important experiment on cat gait by Shik et al. (1966). In this influential experiment, the authors stimulated the mesencephalic locomotor region of a decerebrated cat with increasing levels of electrical currents, applied through an electrode inserted directly in the cat's midbrain. As a result, the authors observed that the cat produced different gaits as a function of the level of stimulation applied, with increasing levels of stimulation eliciting transitions between first a *walk*, then a *trot* and finally a *gallop* gait.

To keep exposition simple, we here ignore the *trot* gait and model a transition between a *walk* and a *gallop* gait depending on a "low" vs "high" level of stimulation. In order to characterize gaits by their patterns, the literature on mammalian quadruped gaits traditionally enumerates the legs of the animal, such that each gait is defined by a sequence of numbers describing the order in which the legs touch the ground. The traditionally adopted leg enumeration is reported in Figure 4.9.

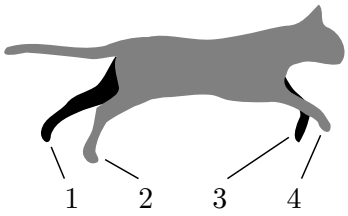


FIGURE 4.9: **Leg enumeration in the study of mammalian gaits.** Through this enumeration, a gait can be represented by the sequence with which the legs touch the ground, starting from leg 1.

The gait pattern is normally reported starting with leg 1 (i.e. the left hind leg), such that a *walk* gait is characterized by the sequence $(1, 3, 2, 4)$, and a *gallop* gait by the sequence $(1, 2, 3, 4)$. The production of these patterns can be easily abstracted through a FSM. Specifically, as we want the produced pattern to depend on a "stimulation level" control parameter, we construct the FSM to produce one of the two patterns depending on which of two input symbols is present, i.e. a `<lo>` and a `<hi>` input symbols. We model the

Symbols	States			
	q_1	q_2	q_3	q_4
<lo>	q_3	q_4	q_2	q_1
<hi>	q_2	q_3	q_4	q_1

TABLE 4.2: **State transition table for the simulated Finite-State Automaton generating gait patterns through state switching.**

production of the gait patterns as the continue transition between a sequence of internal states. The complete transition table defining the FSM is thus reported in Table 4.2.

Having constructed the FSM, we can now use our methodology to construct a RNN simulating it in real-time. We first map the FSM to a VS and define the Gödelizations of the dotted sequences encoding FSM configurations, making it possible to derive a NDA simulating the FSM in real-time. Specifically, the encodings are defined as in Equation 3.10, with the enumerating functions specified as

$$\gamma_s(\sigma) = \begin{cases} 0 & \text{if } \sigma = \texttt{<lo>} \\ 1 & \text{if } \sigma = \texttt{<hi>} \end{cases} \qquad \gamma_q(q) = \begin{cases} 0 & \text{if } q = q_1 \\ 1 & \text{if } q = q_2 \\ 2 & \text{if } q = q_3 \\ 3 & \text{if } q = q_4 \end{cases}.$$

From the NDA, it is possible to derive a RNN simulating it in real-time with 24 units, as shown in Section 4.4. Specifically, the set of states $Q = \{q_1, q_2, q_3, q_4\}$ and the set of input symbols $\mathbf{T} = \{\texttt{<lo>}, \texttt{<hi>}\}$ of the FSM induces a BSL layer with respectively $|Q| = 4$ and $|\mathbf{T}| = 2$ units in its b_x and b_y sub-assemblies, and a LTL layer with $2|Q||\mathbf{T}| = 16$ units. The LTL of the network contains two units for each entry in the transition table in Table 4.2, which receive connections from the BSL layer with the lateral inhibition pattern discussed in Section 4.4.2, such that each pair of units in the LTL is only activated when the BSL selects it. A pair in the LTL is selectively de-inhibited by the BSL only if the transformation it applies corresponds to the transition rule which is to be applied given the contents of the current encoded FSM configuration. That is, each pair in the LTL corresponds to a pair q, σ of current state q and current input symbol σ of the FSM, and is only activated

when the MCL encodes a FSM configuration with q and σ as current state and symbol. When a pair is activated, each unit in the pair applies a linear transformation to one of the units in the MCL layer, updating its activation through a recurrent connection. This action simulates by design the symbolic updating of the FSM configuration (as discussed in 4.4.3), which is now encoded in the new MCL activation.

In an experiment to test the pattern-generation capabilities of the network, we continuously manipulate the activation of the c_y unit (which encodes the input tape of the simulated FSM, and thus the "level of stimulation") to qualitatively reproduce the methods in Shik et al. (1966). Interestingly, we are here introducing a continuous control parameter into what was originally a purely symbolic form of computation. This allows us to carry out bifurcation studies by parameter manipulation, as traditionally done for coupled oscillator model (see Collins and Richmond, 1994, Golubitsky et al., 1998, 1999, Schöner et al., 1990). We report the dynamics of the pattern-generating RNN for increasing levels of input stimulation in Figure 4.10. Note how increasing the levels of stimulation elicits the transition from a *walk* to a *gallop* produced gait as transparently encoded in a spatial-temporal pattern of the network activation. This is made possible by letting the order in the γ_s enumeration be consistent with the semantic order of the two `<lo>` and `<hi>` input symbols. That is, we define γ_s such that $\gamma_s(\texttt{<lo>}) < \gamma_s(\texttt{<hi>})$.

In particular, note in Figure 4.11 how each pair in the LTL is selectively activated by the BSL at each time step, and how the sequence of selective activations reproduces the sequence of transitions in the simulated FSM.

To summarize, we have constructed a 24-units RNN network from a FSM designed to produce two patterns (walk and gallop) depending on the symbols in its input string, as a high-level abstraction of a CPG. In particular, the RNN network produces one of the two rhythmic patterns depending on the activation value of the c_y input unit in the MCL layer, which we experimentally manipulated. The rhythmic pattern is here encoded in the sequence of activations of the units in the LTL layer of the network (as shown in Figure 4.11), which can thus here be considered as the output of the network. As the network is

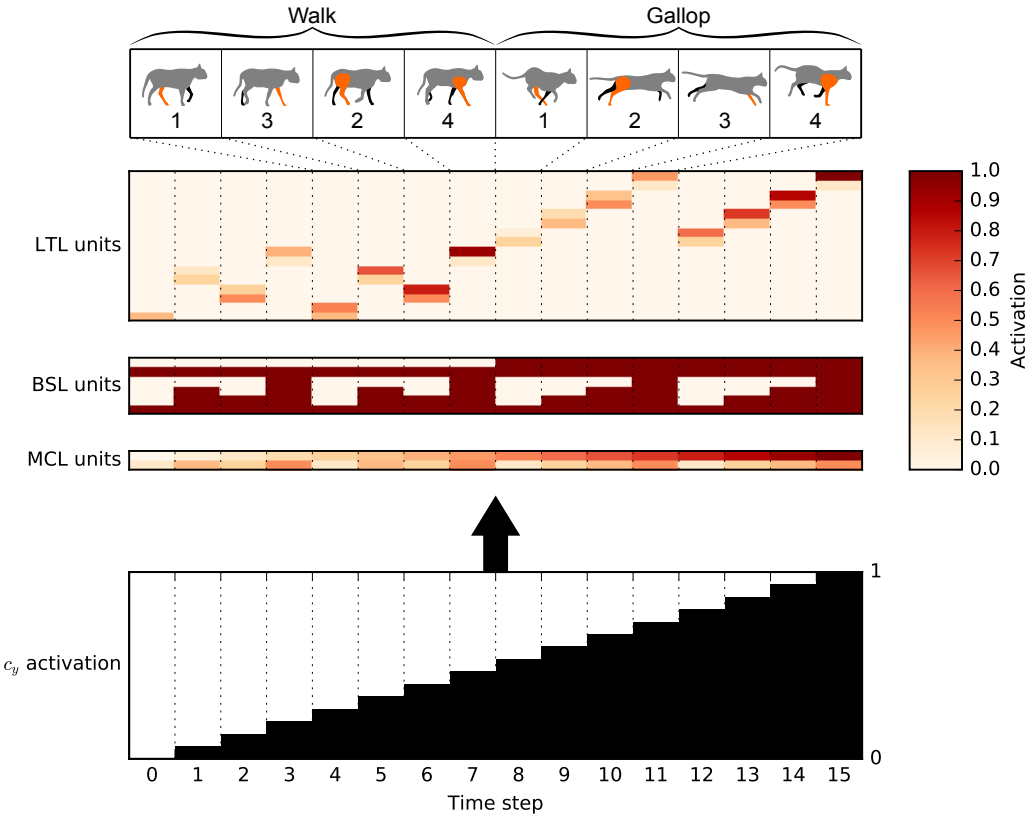


FIGURE 4.10: **Network activation for Recurrent Neural Network functioning as a cat gait Central Pattern Generator**. In the experiment, we manipulate the activation of the MCL c_y unit encoding the level of input stimulation. Note how an increasing level of stimulation leads to a transition between a produced *walk* gait and a *gallop* gait, transparently encoded through a spatial-temporal pattern in the network LTL.

able to produce a rhythmic pattern in the absence of rhythmic input, it can be considered a very simple form of CPG.

By constructing a discrete-time CPG (implemented as a RNN) from an appropriately designed FSM, we essentially outlined a methodology for the design of CPGs that does not suffer from some of the drawbacks of these models, while benefiting from the advantages given by FSM locomotive controllers. This opens exciting possibilities for future research and applications in robot locomotion. However, some issues must be tackled in order for this methodology to provide the full range of advantages of CPGs as summarized at the beginning of this Section. Specifically, many of the benefits of CPGs for robot locomotion can only be ascribed to continuous-time CPGs. Importantly, methods similar to those proposed in Ashwin and Postlethwaite (2013) and Horchler et al. (2015) could be used to construct such CPGs from Finite-State descriptions of the desired pattern generation.

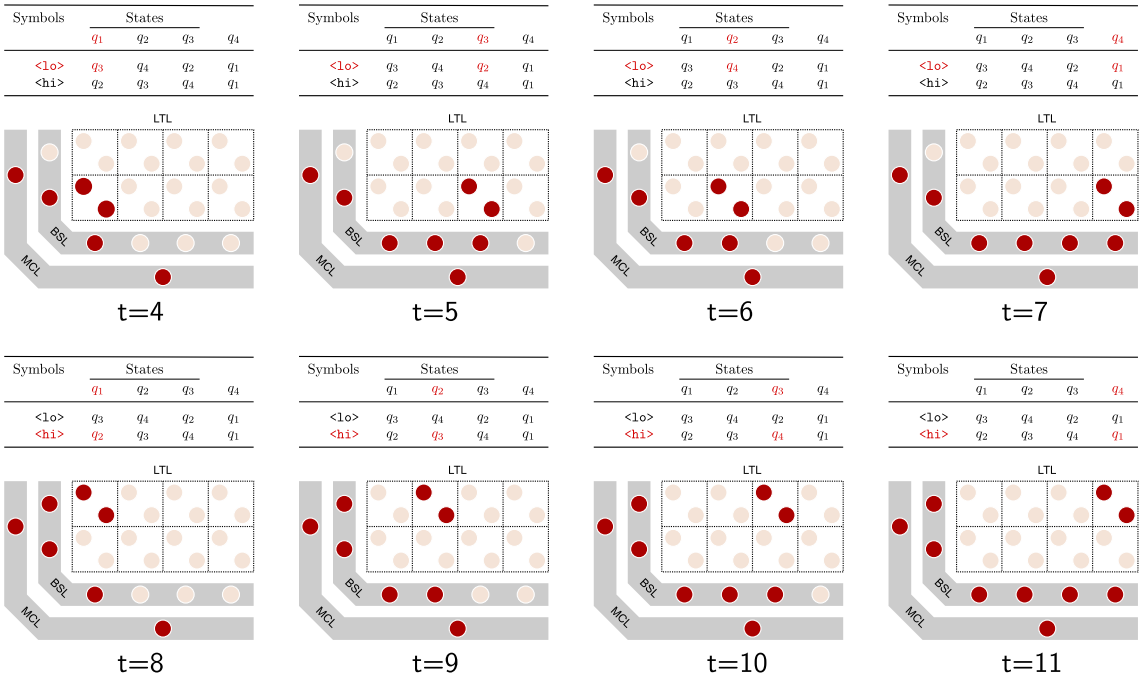


FIGURE 4.11: **Active units in the network for simulation time t = 4...11 (compare with Figure 4.10).** In this figure, each of the units in the network architecture (shown reproducing the visual layout presented in Figure 4.5) is highlighted for a given time step if its activation at that time step is greater than zero. Additionally, the current state, symbol an new state (which can be decoded from the MCL activation in the RNN) are highlighted in the transition table of the FSM for each time step. The upper row shows the RNN activation sequence and FSM transitions for a *walk* gait (corresponding to time steps 4 to 7 in the complete simulation), whereas the lower row shows the RNN sequence and FSM transitions for a *gallop* gait (corresponding to time steps 8 to 11 in the complete simulation).

Recent research has uncovered a high degree of hierarchical organization in mammalian respiratory CPGs, with many interacting components that allow for the production of robust and flexible patterns in a range of conditions (see Smith et al., 2007, 2013). Importantly, as we will show in the next Section, our methodology is ideally suited for the construction of RNN simulating networks of automata that interact with each others and the external environment. For this reason, we believe the present work introduces a unique view on the construction of hierarchically organized CPGs, which are especially relevant once again in the realm of robotics locomotion, especially in the case of modular robots (see Spröwitz et al., 2014 for an example of CPG-controlled modular robot).

4.5.3 Garden-path parser

In this example, we will demonstrate the suitability of our approach for the construction of cognitive models, given in particular by the possibility of tapping the power of interactive computation. Whereas the abstraction underlying the classical approach to computation through the theory of automata and formal languages is that of computation as the intervening step between some initial input and a final output, in the framework of interactive computation (Wegner, 1998) a model of computation can interact with the external world at any time. This is of course a much more powerful approach to the modelling of computation, providing a rich language to express notions of compositionality and concurrency that are of great importance in the description of computation in a range of systems. Interaction with an external environment is, in fact, a hallmark of cognitive systems; as such, by supporting the implementation of interactive systems in RNNs through our approach, we are able to construct transparent connectionist cognitive architectures.

In what follows, we will construct a cognitively inspired parser for locally ambiguous sentences (garden-path sentences, discussed in Section 4.5.3.1). The parser is composed by distinct interactive automata, each one playing a specific role in the overall processing of incoming sentences, and implementing different forms of computation and interaction between each other, demonstrating the strengths of our proposed approach.

4.5.3.1 Garden-path sentences

Consider the following sentence:

The dog that I really had loved bones.

Upon reading the sentence, a reader sequentially constructs a parse with "*The dog*" as the subject of the phrase, and "*that I really had loved*" as a relative clause. In encountering the last word "*bones*", however, the reader becomes aware that their initial parse was incorrect. In fact, "*loved*" was actually part of the main clause "*The dog ... loved bones*" and not of a "*that I really had loved*" relative clause which the reader incorrectly constructed. The

reader thus reanalyzes the sentence to produce the correct parse. This is an example of garden-path sentence, i.e. a sentence containing a local ambiguity such that a reader sequentially processing the sentence is led to the construction of an incorrect parse, which is then unveiled as incorrect by later material in the sentence, forcing the reader into a re-processing of the sentence. This process of reanalysis is associated in the brain of the reader with a positive deflection in potential after 600 milliseconds (P600) from the onset of a garden-path ("*bones*" in the above sentence), as shown by Osterhout et al. (1994) by measuring the trial averaged electroencephalogram during the sequential presentation of garden-path and control sentences.

In particular, Frisch et al. (2004) have conducted Event-Related Potential (ERP) studies on German speakers processing garden-path sentences that arise from subject-object ambiguities. In German, as in a number of other languages, when a nominal constituent is ambiguous in regard to its object/subject role, native speakers prefer to interpret it as a subject rather than an object.

Let us introduce two example sentences, extracted from the ERP study by Frisch et al. (2004). The two sentences are both locally ambiguous. However, while the first sentence is compatible with the preferred strategy of assigning the "subject" role to an ambiguous nominal constituent, the second sentence does not reflect the preferred order, and thus causes a temporary incorrect parse which must be resolved (a garden-path). Specifically, we present the first sentence as

Nachdem	*die Kommissarin*	*den Detektiv*	*getroffen hatte,*	*sah*	*sie*$_{\mathbf{s}}$	*den Schmuggler*$_{\mathbf{o}}$
After	the cop	the detective	had met	saw	she	the smuggler

" After the cop had met the detective, she saw the smuggler"

For this first sentence, the human parser correctly interprets the ambiguous "sie" as the subject of the second clause (as per the preferred strategy) in the nominative case, and thus then correctly interprets "den Schmuggler" as the object in the accusative case, successfully completing the parse.

The second sentence is instead formulated in the dispreferred object-subject (o-s) order, eliciting a garden-path in parsing:

Nachdem	*die Kommissarin*	*den Detektiv*	*getroffen hatte,*	*sah*	*sie*$_{\mathbf{o}}$	*der Schmuggler*$_{\mathbf{s}}$
After	the cop	the detective	had met	saw	she	the smuggler

" After the cop had met the detective, the smuggler saw her"

As in the case of the first sentence, upon encountering "sie" the human parser initially constructs a parse with it as a nominative pronoun and thus the subject of the main clause, following the preferred strategy. At this point, the parser expects an object in the accusative case to appear shortly thereafter. This time, however, the ambiguous pronoun actually defines the object of the clause, as the parser quickly realizes upon encountering "der Schmuggler". In fact "der Schmuggler" is unambiguously declensed in the nominative case, and thus must be the subject of the clause. The human parser has to revise their parse, assigning to "sie" the role of object of the verb "sah", and finally correctly re-interpreting "sie" as a pronoun in the accusative case. As shown by Frisch et al. (2004), this reanalysis is reflected in the ERP by a P600 effect.

The reanalysis of incorrect partial parses in sentences presenting garden-paths has received much attention in the literature, and many models have been proposed to account for its underlying mechanisms. In what follows, we will essentially adhere to a *diagnosis and repair* account of reanalysis, as described in Lewis (1998), to construct a parser that can process toy sentences presenting subject/object ambiguities. In this model, when the parser gets stuck in a garden-path, a need for reanalysis is diagnosed, and the parse is repaired by some *repair operator* that re-places the parse in another point of the search space, such that the parser is able to continue its processing of the input sentence and finally produce a correct and complete parse.

4.5.3.2 Constructing the neural parser

To model the sentences with subject/object ambiguities, we resort to a very high level of abstraction (as in beim Graben et al., 2004), where two possible sentence structures are

defined through a CFG G with non-terminals $\mathbf{N} = \{\mathtt{S}\}$, terminals $\mathbf{T} = \{\mathtt{s}, \mathtt{o}\}$ (standing respectively for "subject" and "object"), starting symbol $\mathtt{S}$ and with the set of production rules R containing two rules

$$\mathtt{S} \rightarrow \mathtt{s}\ \mathtt{o} \qquad (s\text{-}o \text{ sentence})$$

$$\mathtt{S} \rightarrow \mathtt{o}\ \mathtt{s}. \qquad (o\text{-}s \text{ sentence})$$

The reason why we decide to use such a simple sentence model is that in this example we want to focus on the interaction between the components in the parser and the overall computation performed, rather than the intricacies of complex – although more detailed and accurate – grammars.

The parser we construct (which structure is shown in its entirety in Figure 4.12) has two sub-modules specialized in the selective recognition of the two types of sentences, in the form of two TDRs. These are defined by splitting the G grammar in two $G_{s\text{-}o}$ and $G_{o\text{-}s}$, each one comprising of one of the two production rules in R, and deriving their transition function through the methods in Section 1.1.6. In this way, the *s-o* TDR can recognize sentences where the subject/object pronoun ambiguity can be correctly resolved with the preferred strategy (i.e. interpreting the ambiguous pronoun as the subject), whereas the *o-s* TDR is able to recognize sentences that are formed such that the ambiguous pronoun is actually the object of the sentence. The parser first tries to parse its input through the *s-o* TDR, consistently with the preferred strategy observed in native speakers. If the input sentence is not in the subject-object, then parser becomes stuck in a garden-path, as the *s-o* TDR is not able to successfully recognize the sentence. In this case, as our parser adheres to a *diagnosis and repair* account of reanalysis, it first diagnoses that the parse has become stuck, repairs it, and finally completes the parse by using the *o-s* TDR on the remaining input.

The diagnosis step is implemented through the action of a *Diagnosis* PDA, which uses its stack memory to keep track of the parse at each time step, such that it is able to compare the current parse with that of the previous computation step. If the current parse is the same as that of the last step, it means that no progress was made in parsing and thus the parser has become stuck in a garden-path. In this case the *Diagnosis* PDA is able to

Symbols	States		
	${}^{\text{pda}}q_{\text{idle}}$	${}^{\text{pda}}q_{\text{parsing}}$	${}^{\text{pda}}q_{\text{error}}$
$(\sqcup, \sqcup)$	$({}^{\text{pda}}q_{\text{idle}}\ ,\ \sqcup)$	$({}^{\text{pda}}q_{\text{idle}}\ ,\ \sqcup)$	$({}^{\text{pda}}q_{\text{idle}}\ ,\ \sqcup)$
(x, y)	$({}^{\text{pda}}q_{\text{parsing}}\ ,\ y)$	$({}^{\text{pda}}q_{\text{parsing}}\ ,\ y)$	$({}^{\text{pda}}q_{\text{parsing}}\ ,\ y)$
$(x, x);\ x \neq \sqcup$	$({}^{\text{pda}}q_{\text{error}}\ ,\ x)$	$({}^{\text{pda}}q_{\text{error}}\ ,\ x)$	$({}^{\text{pda}}q_{\text{error}}\ ,\ x)$

TABLE 4.3: **State transition table for the *Diagnosis* Push-Down Automaton.** The automaton pushes its input at the top of the stack for every state and input, after comparing the current input with its top-of-stack (i.e. the input stored from the previous time step). If the input and top-of-stack are the same, then the parser is stuck, and the automaton switches to an error state ${}^{\text{pda}}q_{\text{error}}$. Otherwise, it stays in a parsing state ${}^{\text{pda}}q_{\text{parsing}}$. If no input is present at the current and preceding time step, instead, the automaton switches to an idle state ${}^{\text{pda}}q_{\text{idle}}$. This diagnostic information is used by the *Strategy* Finite-State Machine to control the parsing strategies to apply on the input.

diagnose the problem and signal it to an upper control system by switching its state to an "error" state. The transition function for the *Diagnosis* PDA is reported in Table 4.3.

The repair operation is implemented through a *Repair* VS, which modifies the current parse, allowing the parser to continue its computation. The VS can be defined by the rewriting rule

$$\texttt{s o} \,.\, w \rightarrow \texttt{o s} \,.\, w, \tag{4.24}$$

which corresponds to a reanalysis of the input sentence, such that it is interpreted through the dispreferred object-subject sentence structure. This makes it possible for the *o-s* TDR to take over and thus correctly recognize the input sentence.

We have introduced four of the five basic components of the parser, i.e. three parsing automata (the *s-o* and *o-s* TDR and the *Repair* VS), and the *Diagnose* PDA. The last component left to describe is the *Strategy* FSM, a high-level controller that receives diagnostic information from the *Diagnosis* PDA as input, and thus decides which parsing strategy to apply to the input at each time step by selectively activating the corresponding automaton. Note that this form of interaction, i.e. the "function call" of an automaton from another automaton, however intuitive from the point of view of its logic, was not defined for the VS introduced in Section 4.1. In fact, we did not endow VS with the capability of calling other VSs. While on the one hand we leave the implementation of such capabilities for future work, to best adapt the VS object to the interactive computation

Symbols	States		
	${}^{\text{fsm}}q_{\text{s-o}}$	${}^{\text{fsm}}q_{\text{o-s}}$	${}^{\text{fsm}}q_{\text{repair}}$
${}^{\text{pda}}q_{\text{idle}}$	${}^{\text{fsm}}q_{\text{s-o}}$	${}^{\text{fsm}}q_{\text{s-o}}$	${}^{\text{fsm}}q_{\text{s-o}}$
${}^{\text{pda}}q_{\text{parsing}}$	${}^{\text{fsm}}q_{\text{s-o}}$	${}^{\text{fsm}}q_{\text{o-s}}$	${}^{\text{fsm}}q_{\text{o-s}}$
${}^{\text{pda}}q_{\text{error}}$	${}^{\text{fsm}}q_{\text{repair}}$	${}^{\text{fsm}}q_{\text{o-s}}$	${}^{\text{fsm}}q_{\text{o-s}}$

TABLE 4.4: **State transition table for the *Strategy* Finite-State Machine.** The *Strategy* Finite-State Machine selectively activates the three *s-o*, *o-s* and *Repair* automata for the parsing of the input, depending on the diagnostic information received from the *Diagnosis* Push-Down Automaton. The machine start in state ${}^{\text{fsm}}q_{\text{s-o}}$, applying the preferred *s-o* parsing strategy. If the input sentence does not respect the preferred structure, the parsing gets stuck in a garden-path, and thus the *Diagnosis* Push-Down Automaton signals an error by switching to a ${}^{\text{pda}}q_{\text{error}}$ error state. In this case, the *Strategy* automaton first switches to a ${}^{\text{fsm}}q_{\text{repair}}$ state, activating the *Repair* Versatile Shift, which repairs the parse. As a result, the *Diagnosis* automaton diagnoses that the parsing is restored by switching to a ${}^{\text{pda}}q_{\text{parsing}}$ state, thus leading the *Strategy* Finite-State Machine to transition to a ${}^{\text{fsm}}q_{\text{o-s}}$ state, activating the *o-s* Top-Down Recognizer which can finally complete the parsing of the input sentence.

framework (and extend the mapping with RNN to incorporate the new possibilities), on the other hand we are eager to demonstrate the power of our approach and show how the network architecture we introduce can easily accommodate these capabilities thanks to its modularity and transparency. We will thus ignore, for the moment, the missing link with VSs and implement the "function call" capability in the RNN by employing the same lateral inhibition mechanism which we already described in Section 4.4.2 while describing the BSL in the network. We define the FSM such that, when a sentence is first presented, it activates the *s-o* TDR, associated with the preferred parsing strategy. If the sentence is not in the subject-object order, then the defined *Strategy* FSM reads the error signalled by the *Diagnosis* PDA, and thus first activates the *Repair* VS in order to reanalyze the input sentence, and then the *o-s* TDR to complete the parsing. The FSM selectively activates each of the three parsing automata by switching to one of its three states, i.e. a "*s-o*" state, a "*o-s*" state and a "*repair*" state, as prescribed by its transition function, reported in Table 4.4. By selectively activating one between the three parsing automata at any given time, the *Strategy* FSM also makes sure that race conditions are avoided in the rewriting of the "input" and "parse" sub-sequences by these automata. At the same time, the *Strategy* FSM can only read, but not rewrite, the "diagnosis" sub-sequence, and the *Diagnosis* PDA can only read, but not rewrite, the "parse" subsequence. In this way at most one automaton

in the interactive network has rewriting access to a given sub-sequence at any given time, thus avoiding race conditions for all the sub-sequences. The complete parser is shown in Figure 4.12.

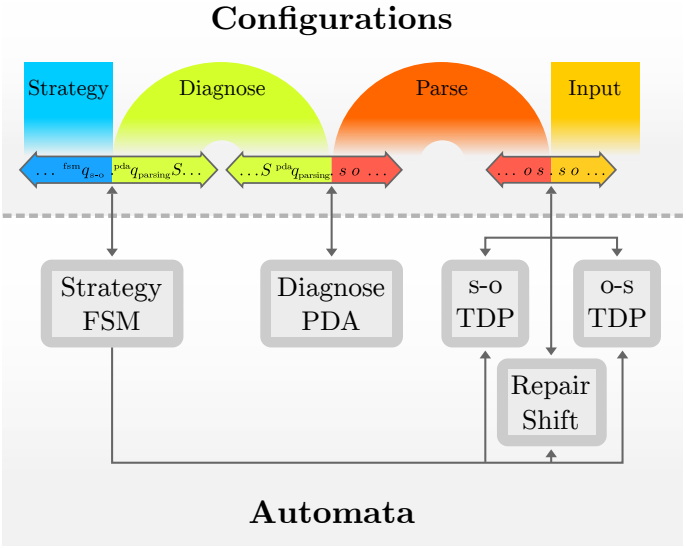


FIGURE 4.12: **Diagram of Interactive Automata Network for parsing of garden-path sentences.** The bottom panel shows the components of the automata network, which communicate through networked access to the relevant parts of each other's configurations, shown in the top panel as dotted sequences. Note how the *Strategy* Finite-State Machine is endowed with an additional form of interaction, selectively activating the *s-o* and *o-s* Top-Down Recognizers, and the *Repair* Shift, depending on its current state.

We will now map the parser to a RNN, by first representing each of its automaton as VSs acting on dotted sequences (by the methods discussed in Section 4.2), mapping each of the obtained VS to a NDA simulating its computation (as discussed in Section 4.3), each NDA to the simulating RNN (as in Section 4.4), and finally, by appropriately connecting the obtained sub-networks allowing for their interaction. In particular, the last step is made trivial thanks to the transparency and modularity of the presented architecture.

To map VSs to NDA, we first need to define the appropriate Gödelizations, as shown in 4.3. Hence, we define the Gödelizations of the "input", "parse" and "strategy" sub-sequences as in Equation 3.10, whereas we define the Gödelization of the "diagnosis" subsequence as in Equation 4.12. The four γ functions enumerating the symbols in the four sub-sequences

are specified to be

$$\gamma_{\text{input}}(x) = \begin{cases} 0 & \text{if } x = \sqcup \\ 1 & \text{if } x = \texttt{S} \\ 2 & \text{if } x = \texttt{o} \\ 3 & \text{if } x = \texttt{s} \end{cases}, \qquad \gamma_{\text{parse}}(x) = \begin{cases} 0 & \text{if } x = \sqcup \\ 1 & \text{if } x = \texttt{o} \\ 2 & \text{if } x = \texttt{s} \end{cases}, \tag{4.25}$$

$$\gamma_{\text{diagnosis}}(x) = \begin{cases} 0 & \text{if } x = {}^{\text{pda}}q_{\text{idle}} \\ 1 & \text{if } x = {}^{\text{pda}}q_{\text{parsing}} \\ 2 & \text{if } x = {}^{\text{pda}}q_{\text{error}} \end{cases}, \quad \gamma_{\text{strategy}}(x) = \begin{cases} 0 & \text{if } x = {}^{\text{fsm}}q_{\text{s-o}} \\ 1 & \text{if } x = {}^{\text{fsm}}q_{\text{o-s}} \\ 2 & \text{if } x = {}^{\text{fsm}}q_{\text{repair}} \end{cases}.$$

Specifying the Gödelizations for the dotted sequences allows us to construct NDA from the VS, and thus derive the RNNs simulating the original models of computation in real-time.

Each of the derived RNNs constitutes a component of a bigger recurrent network, which is shown in Figure 4.13. In order to facilitate exposition, we construct the overall architecture to contain 4 intermediate "Configuration Layers" (CLs), each one comprising of 4 units, encoding respectively the "strategy", "diagnosis", "parse" and "input" sub-sequences. In this way, the parsing sub-networks (i.e. the *s-o*, *o-s* and *repair* sub-networks) receive input from the "parse" and "input" units of CL 1, and send their updated encoded configuration to the corresponding units in CL 2; the *Diagnosis* sub-network, instead, receives the encoded current "diagnosis" and "parse" from the corresponding units in CL 2, and outputs the result of its computation to the corresponding units CL 3; finally, the *Strategy* sub-network receives input from the "strategy" and "diagnosis" units in CL 3 and outputs to the corresponding units in CL 4, which is recurrently connected to CL 1 with connection matrix $\mathcal{I}_{4\times4}$. Additionally, a *Meta* BSL is added to the network to implement the "function call" capability of the *Strategy* FSM. This layer is constructed as a single b_x BSL group of three units, taking input from the "strategy" unit in CL 1 and selectively activating one of the parsing sub-networks through the pattern of lateral inhibition discussed in Section 4.4.2. Interestingly, this creates a kind of nested structure, with each of the three parsing sub-networks acting like a single operation of a higher-level symbolic automaton, and thus

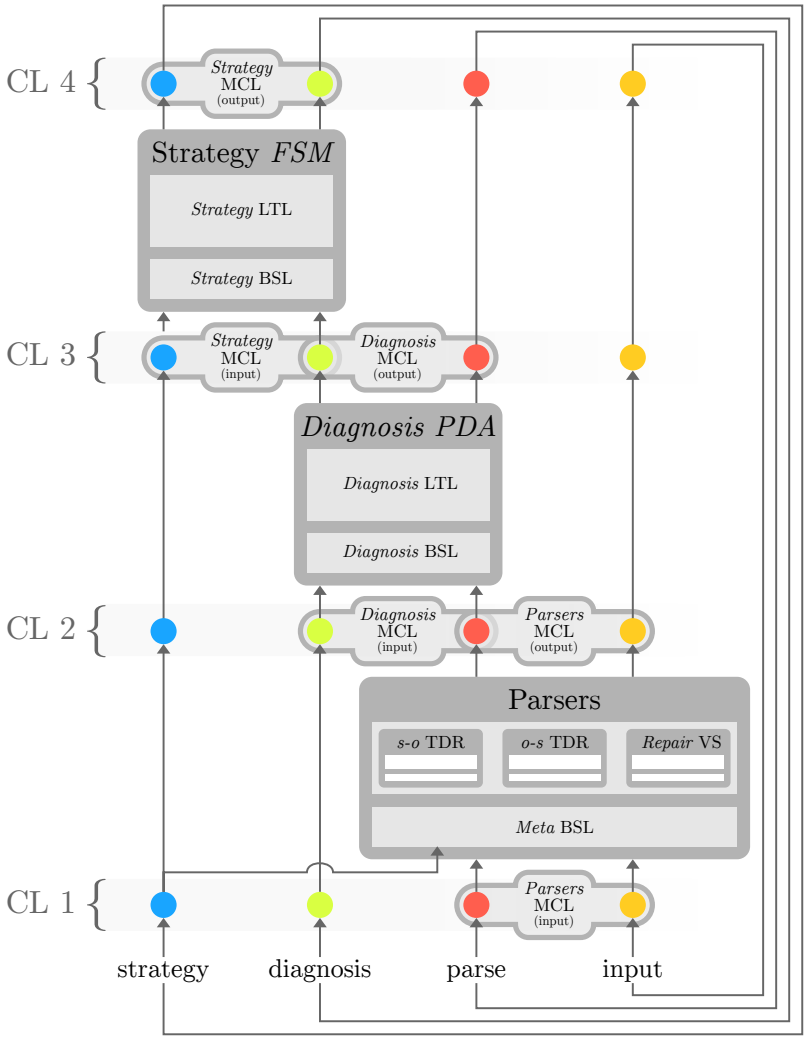


FIGURE 4.13: **Garden-path parsing network architecture.** In this example, we construct the network to only contain one set of recurrent connections, from Configuration Layer (CL) 4 to 1, to simplify exposition. The sub-networks corresponding to the various automata are shown. In particular, note how the *Parser* sub-network is itself composed by the *s-o*, *o-s* and *Repair* sub-networks, each simulating their corresponding automata from the original interactive system.

as cells in a NDA, or pair in a LTL.

4.5.3.3 Results

We have constructed a RNN simulating a network of interactive automata in real-time with 265 units (see Equation 4.23), performing the parsing of toy input sentences modelling pronominal subject-object ambiguity. We are now able to observe this computation in the

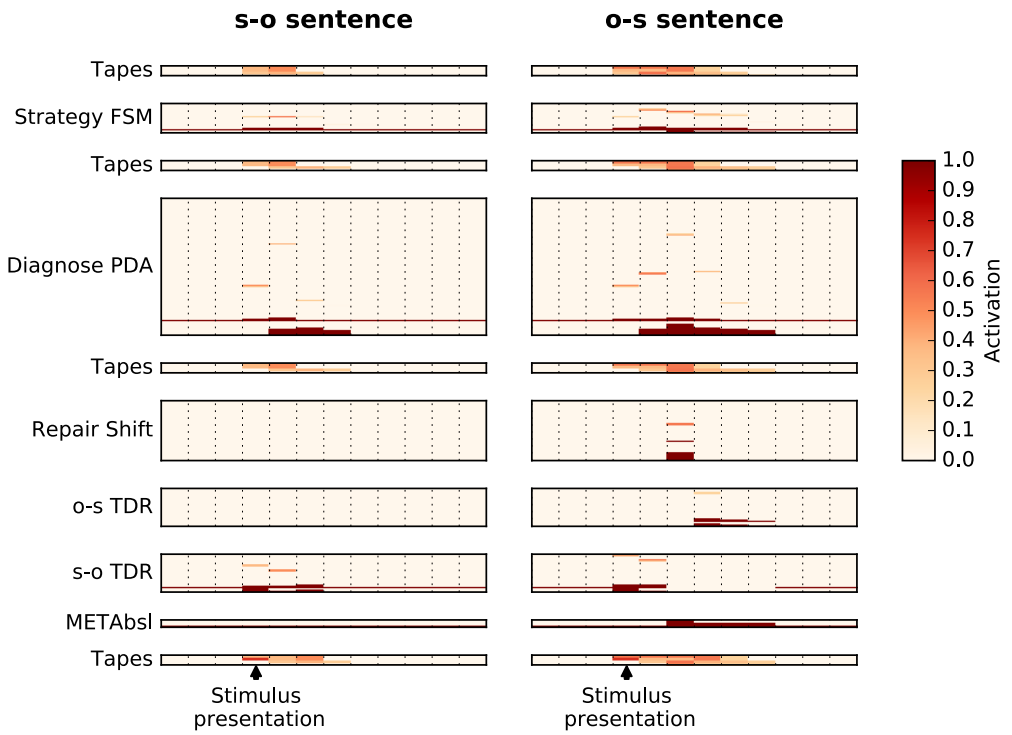


FIGURE 4.14: **Network activation for subject-object and object-subject input sentence.** The activation in time of each unit is reported for a presented input encoding an *s-o* sentence (on the left) and a *o-s* sentence (on the right). In particular, the *o-s* sentence elicits a diagnosis-and-repair operation in the neural parser, which can be observed in the form of a switching between the *s-o*, *o-s* and *Repair* sub-networks, and by the longer tail of activation reflecting the additional computational payload in the processing of the dispreferred sentence structure.

form of neural dynamics in the network, as shown in 4.14, where we report the activation in time for each unit when encoded input sentences with subject-object and object-subject structure are presented.

In particular, the Figure shows the sequential activation of the three parsing networks in the case of the sentence in the object-subject order, consistently with the diagnose-and-repair mechanism discussed in the previous Sections.

Interestingly, by mapping the interactive networks symbolic dynamics to neural dynamics defined on a vector space, we can extract interesting measures that were not defined for the original symbolic model of computation. For example, we can reduce the high-dimensional neural dynamics to a lower-dimensional space, allowing for the comparison with experimental measures from neurophysiology or psychology, such as ERP, EEG, and Local Field Potentials (LFP) measures. The possibility of performing such comparisons by developing

appropriate biophysically-inspired observation models (such as synthetic ERPs, LFPs, or EEGs) has received attention in recent years in the field of computational neuroscience (see Barrès et al., 2013, beim Graben and Rodrigues, 2013).

Examples of standard techniques that can be employed to perform the dimensionality reduction and thus derive synthetic ERPs from the dynamics of our network are Principal Component Analysis (PCA), as shown in beim Graben et al. (2008), Smolensy's harmony (Smolensky, 1986), defined as $H = \sum_{ij} u_i w_{ij} u_j$ with $\boldsymbol{u} = (u_i)$ being the network activation vector and $\boldsymbol{W} = (w_{ij})$ being the synaptic weight matrix, and Amari's mean network activity (Amari, 1974), simply defined as the arithmetic mean of the neural activation over all neurons in the network, i.e. as

$$A = \frac{1}{n} \sum_i u_i \, . \tag{4.26}$$

To extract the synthetic ERPs, we will here use Amari's measure to compute the mean global network activation for the parser RNN, and average it over a number of trial with "noisy" input stimuli. Specifically, we run two different simulations, with input stimuli being respectively the encoded sequences "$\ldots(\sqcup)$`S.so`$(\sqcup)\ldots$" and "$\ldots(\sqcup)$`S.os`$(\sqcup)\ldots$", where $(\sqcup)$ denotes an infinite padding of the blank symbol. For each simulation, we run 100 trials, where the input stimulus is presented at time $t = 2$, and prepared as in beim Graben et al. (2008), where random noise is added to the input stimulus at each trial. The strength of the added random noise is such that the initial dotted words `S.so` and `S.os` can still be recovered correctly by decoding the input, but with the rest of the decoded sequence being a random symbolic continuation. This is done to avoid noise destroying key information relative to the computation to be performed by the network. To extract the synthetic ERPs from each of the two simulations, we compute the network mean activation for each time step, and average it over the 100 presented input stimuli. The results of these simulations are shown in Figure 4.15. In particular, the Figure shows a P600-like effect in the processing of garden-path sentences by the network, where a garden-path sentence elicits a significant peak in activation, sustained until the end of the computation. While the toy model of garden-path processing used in this example is not detailed enough to allow for a quantitative comparison with real ERP data from experiments such as Frisch et al. (2004),

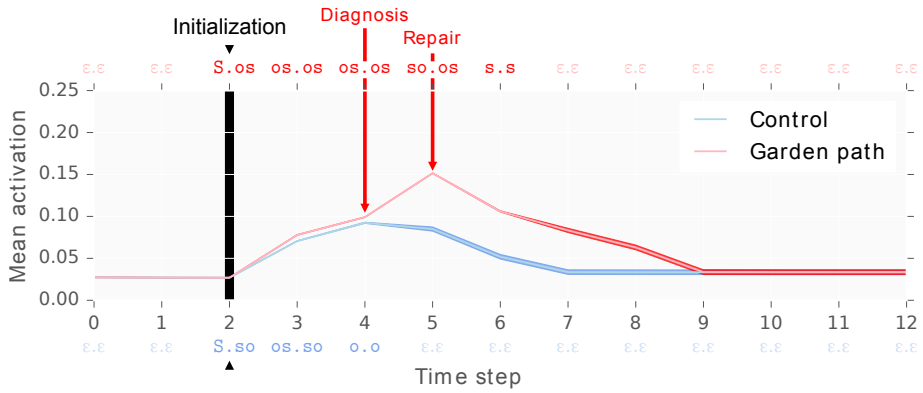


FIGURE 4.15: **Synthetic Event-Related brain Potential for random cloud of initial conditions.** We simulate the network dynamics for two sets of noisy encoded input sentences (refer to Section 4.5.3.3 for details). The first set (control condition) consists of 100 encoded input sentences in the preferred subject-object form, while the second (garden-path condition) consists of 100 encoded input sentences in the dispreferred object-subject form. We report the mean of the average network activation (see Equation 4.26) at each computation step for the control condition (in light blue) and the garden-path condition (in light red), as well as its standard deviation (respectively in dark blue and dark red). We also present, for each computation step, the corresponding state of the parse (the dotted sequence composed of the *parse* and *input* one-sided sequences) for both conditions. Furthermore, we highlight for the garden-path condition the computation steps corresponding to the *Diagnose* and *Repair* operations. Note how the garden-path condition is associated with a long-lasting increase in activity, peaking at $t = 5$. This is reminiscent of the P600 ERP effect in psycholinguistic garden-path experiments. While the model we constructed is too crude for a direct quantitative comparison with experimental data, these simulations could be the starting point for more sophisticated modelling, allowing correlational analyses with real data from experiments.

the construction of more detailed models will lead, through the methods we outlined here, to the possibility of more detailed correlation analyses with electrophysiological data (see beim Graben and Drenhaus, 2012, Frank et al., 2015).

4.6 Conclusions

In this Chapter we introduced a new methodology to map automata computation to the dynamics of RNNs that simulate the computation in real time. Our construction relies on the introduction of a novel shift map, the *Versatile Shift*, which can be simulated by the dynamics of a NDA evolving on a vectorial space, and the definition of an appropriate RNN architecture which reproduces the NDA structure and dynamics as the time evolution of the activations of the neurons in the network.

The Versatile Shift we introduce is a much more expressive model of computation than the Generalized Shift it extends. Whereas the GS can only rewrite symbols in a predefined region of a dotted sequence with new symbols (such that each symbol in the region is rewritten with a new one), a VS allows for the localized rewriting of sub-sequences with new sub-sequences of arbitrary length (where, for example, a sub-sequence of 3 symbols can be substituted with one of 10). In this way, it is possible to simulate a wider range of models of computation parsimoniously and in real-time. That is, as we discussed in Section 4.2, it is possible for a range of automata to define a one-to-one map from an automaton transition function δ to the update function Ω of some Versatile Shift, such that each computation step in the obtained Versatile Shift is in direct correspondence with a computation step of the automaton it simulates. Importantly, the arbitrary localized substitution of sub-sequences by a Versatile Shift clearly suggests its straightforward application in future work to the simulation of models of computation based on term rewriting, such as Grammars and Calculi (Smullyan, 1961). Thanks to the maps we have introduced in order to map VSs to NDA, and these to RNN, the characteristics of the VS would allow for the simulation of such symbolic rewriting systems as NDA and RNN dynamics on vectorial spaces, in very much the same way we have discussed for the case of automata models of computation. As such, our contribution is wider in scope in respect to previous work and defines a general framework for the simulation of symbolic computation through RNN dynamics. Additionally, as discussed in Section 4.4, our approach differs from previous work in that, thanks to the relationship between the network architecture we present and its defining model of computation (as mapped through the intermediate VS and NDA), the presented network is strongly modular in its structure. It is in fact possible to clearly pinpoint, for each atomic operation defined in the simulated automaton, which part of the network implements its action. This transparency is of particular importance when considering the simulation of Interactive Automata Networks. In fact, as we show in Section 4.5.3, the structure of the architecture we present allows for the straightforward composability of simulating RNNs, where sub-networks implementing specific operations (in the form of automata computation) can be easily wired together to form a larger overall network implementing some higher-level computation, much like different sub-routines can be composed to obtain a program.

It has to be noted that the presented framework also suffers from important drawbacks, mainly stemming from discontinuities in the dynamics of the RNN, which derive from the nature of the encoding used to map symbolic sequences to points in vectorial spaces, i.e. the Gödelization. Specifically, the Gödelization can map sequences that are arbitrarily different in terms of the symbols they contain, to points in the vectorial space that are arbitrarily near each other. This is in fact the reason why NDA are piecewise systems, with a switching rule splitting the unit square in a set of disjoint cells which cover it. The discontinuities that Gödel encodings impose on the NDA dynamics (and thus on the simulating RNN dynamics) lead to two important consequences. The first one is that the dynamics of NDA, and thus of the RNN architecture we define, is extremely sensitive to noise in the encoding of the dotted sequences, and in particular when the encoded dotted sequence lies near to the boundary between cells of the NDA. In this case, in fact, arbitrarily small noise applied to the state of the NDA (the encoded dotted sequence) can cause it to lie within another cell from the one in which the state would have lied in the noiseless case. This displacement destroys all the symbolic information encoded in the NDA state, in that decoding the state leads to a completely different sequence from the one we started with. Of course, this also destroys the computation performed by the NDA, as its behaviour is now completely uncorrelated to its initial encoded symbolic input. As the RNN dynamics simulates the underlying NDA dynamics, the same sensitivity to noise is present in the proposed architecture. The second consequence of the discontinuous dynamics caused by Gödelization, is that gradient-based methods for the training of neural networks do not apply readily to our architecture, as arbitrarily small changes in the weights of the network can cause large differences in the computation it performs. It might be possible to overcome these drawbacks in future work by employing a different class of encodings in which vectorial and symbolic distances correspond more faithfully. A promising candidate is the class of tensor encodings, introduced by Smolensky in 1990. In particular, tensor encodings offer a fully distributed representation which can be used to store structured (possibly recursively structured) data in the pattern of activation of neural units, while preserving a more natural correspondence between euclidean distance in vectorial space and distance in the encoded symbolic data, which is reflected in the absence of the catastrophic effects of noise discussed for Gödel encodings, and in the suitability of the encoding to be

used with gradient descent methods for learning.

Chapter 5

Heteroclinic computation in networks of oscillators

The dynamics of systems of artificial oscillators can support the presence of networks of unstable states (saddles), i.e. heteroclinic networks (see Chapter 2). External inputs applied to the oscillators can then force the dynamics to continuously switch between states (Neves and Timme, 2009, 2012), realizing a computation (Ashwin and Borresen, 2005, Ashwin and Marc, 2005a, Kirst et al., 2009, Krupa, 1997, Timme et al., 2003, Wordsworth and Ashwin, 2008), and giving rise to the paradigm known as *heteroclinic computing*. In what follows, we first characterize the form of computation performed by this class of systems in terms of symbolic dynamics, showing that they are dynamical implementations of Finite State Transducers (Section 5.1). Secondly, we characterize the amount of information they can transmit (Section 5.3). We then introduce a class of systems where the heteroclinic dynamics is given by the transition between synchronized states of oscillator activation (Section 5.2). Finally, we simulate a heteroclinic system exhibiting a network of synchronized states to show that, when input from a noise source is included in the system, it can allow for a greater capacity and speed of information transmission (Section 5.4). This result interestingly hints to the fact that dynamical implementations of symbolic models of computation can support emerging phenomena fundamentally outside what the original

symbolic models can capture, while still falling within an intuitive and intelligible notion of information processing.

5.1 Symbolic Dynamics of heteroclinic networks

To study how heteroclinic networks in the state space of a dynamical system can support the processing of symbolic information, we can apply the theory of symbolic dynamics. The discretization of a dynamical system's dynamics is normally achieved in the symbolic dynamics framework by defining a partition of the state space, such that a sequence of output symbols is induced by the sequence of cells in the partition visited by the dynamical system state. Crucially, in the case of heteroclinic networks, the partition of the state space is naturally induced by the graph structure characterizing the network.

A heteroclinic network can be represented as a directed graph G where the set of vertices $V(G)$ is the set of unstable states (saddles) in the network, and where the edges $E(G)$ are the heteroclinic connections between those states. The transition between states in the noiseless system depends on the sign of the difference of perturbations to the unstable directions of the approached state. That is, the signal component with the largest magnitude in the direction of one of the unstable manifolds of the current saddle forces the dynamics of the system towards that specific unstable direction and, finally, to the associated saddle. For this reason, we can define an input labeling $\mathcal{I}$ of G where each edge e from a vertex v_i to a vertex v_j is labelled corresponding to which signal component to the unstable directions of v_i has to be the strongest in order to drive the system towards v_j (see Figure 5.1)

Through this abstraction, we can give a symbolic interpretation of the dynamics of systems with heteroclinic networks, where input perturbations take the role of input symbols, and the reaching of a state can be interpreted as the production of an output symbol from an alphabet $\mathcal{O} = V(G)$. That is, a heteroclinic network can be seen as implementing a Finite-State Transducer (see Section 1.1.3), allowing the re-encoding of input perturbations into output sequences of states.

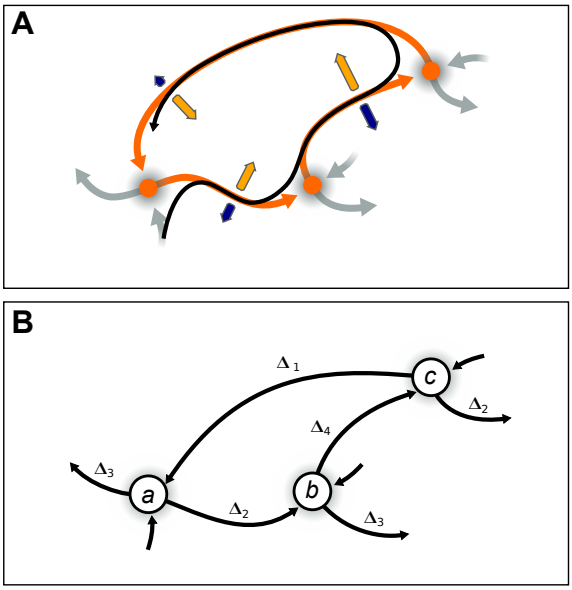


FIGURE 5.1: **Graph representation of heteroclinic network.** (A) A heteroclinic network of 3 saddle states is shown, where at each saddle perturbations to the state of the system drive the dynamics towards the unstable direction corresponding to the largest perturbation component. (B) By seeing each saddle as a discrete state, each heteroclininic connection as a discrete state-to-state vertex, and by labelling each vertex by the signal component that, if strongest, would drive the dynamics towards its associated connection, we can obtain a Finite-State Transducer (see Section 1.1.3) abstraction of the system.

The space of possible bi-infinite heteroclinic sequences defined by a heteroclinic network is the vertex shift $\mathbf{X}_G$ defined on the network graph G, i.e.

$$\mathbf{X}_G = \{x = (x_i)_{i\in\mathbb{Z}} \in \mathcal{O}^{\mathbb{Z}} \mid A_{x_i x_{i+1}} = 1 \quad \text{for all } i \in \mathbb{Z}\} \tag{5.1}$$

where A is the adjacency matrix of the heteroclinic network, with $A_{ij} = 1$ if state i has a heteroclinic connection to j, and $A_{ij} = 0$ otherwise. This shift space defines a language $\mathcal{L}(\mathbf{X}_G) = \bigcup_{n=0}^{\infty} \mathcal{B}_n(\mathbf{X}_G)$, i.e. the set containing all possible words in $\mathbf{X}_G$. Given that the vertex shift $\mathbf{X}_G$ can be described by a finite set of forbidden 2-words (i.e. the set of forbidden state-to-state transitions in the heteroclinic graph), $\mathbf{X}_G$ is a shift of finite type, and thus the language $\mathcal{L}(\mathbf{X}_G)$ is a regular language. This of course should come as no surprise, given how we characterized computation in heteroclinic networks as being that of Finite-State Transducers (the class of automata generating regular languages).

5.2 Heteroclinic networks emerging from systems of oscillators

Let us broadly define a system of N identical oscillators having the form

$$\frac{dV_i}{dt} = F(V_i) + \sum_{j \neq i}^{N} W_{ji}(t) + S_i(t) + \eta_i \tag{5.2}$$

where $F(V_i) : \mathbb{R} \to \mathbb{R}$ is a non-autonomous function of Voltage-like variable $V_i \in \mathbb{R}$ of the i-th oscillator, $\sum_{j \neq i}^{N} W_{ji}(t)$ is input to oscillator i from other oscillators $j = 1, \ldots, N$ with $j \neq i$, $S_i(t)$ an external input and η_i a Gaussian noise source. Here the choice of F is further constrained on the condition that for no external and network input and no noise, the dynamics $\frac{dV_i}{dt}$ of each oscillator is periodic. That is, when $\sum_{j \neq i}^{N} W_{ji}(t) + S_i(t) + \eta_i = 0$ for all t), a period T exists such that $V_i(t + T) = V_i(t)$ for all t. Systems of this form can sometimes support the emergence of heteroclinic networks, and thus function as symbolic computing machines as outlined in the previous Section. Specifically, when exploring the forms of computation these systems can support, it is natural under a Dynamical Systems perspective to characterize their dynamics in state space as the *output* they produce given some external input; in the simplest case, the external input $\boldsymbol{S} = (S_i)$ to the system can be characterized as a vector of fixed detuning currents $\boldsymbol{\Delta} = (\Delta_1, \Delta_2, \ldots, \Delta_N) \in \mathbb{R}^N$, such that $S_i = \Delta_i$.

Wordsworth and Ashwin (2008) and Neves and Timme (2012) have shown that symmetric systems of N oscillators receiving N independent detuning currents can implement a persistent switching of states characterized by the production of cyclic sequences, implementing a k-winner-take-all ($k < N$) computation separating the k stronger from the $N - k$ weaker detuning currents (with k depending on the specific system). That is, they compute a partial ordering of their input currents. We will refer to this class of systems as *State-Switching Machines* (SSMs). In SSMs, the output states of the emerging heteroclinic network correspond to partial synchronizations (clusters) of the oscillator voltages, where perturbations to the voltages can result in the de- and re-synchronization of subsets of oscillators, resulting in the approaching of a new clustered state (see Section 5.4.1 for a

concrete example). The input sequences to the emerging heteroclinic Finite State Transducer are instead induced by the application of the $\mathbf{\Delta}$ vector of detuning currents to the system of oscillators. In the noiseless system ($\eta_i = 0$ for all $i = 1, \ldots, N$), a fixed detuning signal to the oscillators drives the system towards specific unstable directions each time a state is approached. The input symbol at each state is thus just the strongest between the signal components towards the unstable directions, as it determines the next approached state. In other words, at each state state the system performs a comparison between the strength of the perturbations to the unstable directions of its associated saddle (see panel A of Figure 5.1). We are thus able to retrieve a partial ordering of the detuning currents to the oscillators of a SSM by simply observing which output sequence of states it induces.

5.3 Information transmission in noiseless SSMs

We want to characterize the amount of information that can be transmitted by SSMs emerging from systems of oscillators, when they are seen are re-encoders of input currents to the oscillators into output sequences of states. To do so, we will define a discrete input set of vectors of currents for a generic SSM, where each vector is seen as a discrete *symbol*, and compute the Mutual Information (see Sections 1.2.1 and 1.2.2) between these and the output sequences which the machine produces. The input set can be chosen arbitrarily, and many reasonable choices are possible; nevertheless, given that SSMs compute a partial ordering of their input currents, as discussed in the previous Section, a natural choice of input set X is that containing all the possible orderings of the currents of a given ordered $\mathbf{\Delta}$ vector, i.e. the set $\{r(\mathbf{\Delta}) | r \in \mathcal{P}(\mathbf{\Delta})\}$ of all possible permutations of the currents in $\mathbf{\Delta}$ (where $\mathcal{P}$ is the set of all bijections from $\mathbf{\Delta}$ to itself).

For a SSM of N oscillators, the number of possible orderings is thus equal to $|X| = N!$ (as $N!$ is the number of ways n different object can be permuted). The output periodic sequence of states of the noiseless SSM allows the categorization of the N input signals to the oscillators in two groups of size k and $N - k$; the system cannot determine the internal orderings of the two groups. This means that the system can only effectively distinguish between $\binom{N}{k} = \frac{N!}{(N-k)!k!}$ percepts.

To derive the Mutual Information $I(X;Y) = H(X) - H(X|Y)$ between input and output of the SSM (represented by random variables X and Y, see Section 1.2), and thus characterize the amount of information the system can transmit, let us first assume a uniform distribution for the probability of the different X inputs. In this case, it is easy to compute the marginal input entropy as

$$H(X) = -\sum_x p(x) \log p(x) = -\log \frac{1}{N!}. \tag{5.3}$$

where x is one of the possible outcomes of the random variable X representing the input to the system.

For what concerns the conditional entropy $H(X|Y) = -\sum_y p(y) \sum_x p(x|y) \log p(x|y)$, we know that $p(x|y)$ in the noiseless system is equal to 0 if x does not have the partial ordering associated with the sequence y, and equal to $\frac{1}{(N-k)!k!}$ if it has (as $(N-k)!k!$ inputs share the same partial ordering). This leads to a conditional entropy of

$$H(X|Y) = -\log \frac{1}{(N-k)!k!}. \tag{5.4}$$

and, finally to a Mutual Information of

$$\begin{aligned} I(X;Y) &= -\log \frac{1}{N!} + \log \frac{1}{(N-k)!k!} \\ &= \log \frac{N!}{(N-k)!k!} \\ &= \log \binom{N}{k}. \end{aligned} \tag{5.5}$$

5.4 Information transmission in noisy SSMs

When noise is present in SSMs, the sequences of states produced by the heteroclinic switching are not periodic anymore, as the stochastic effects of noise makes the switching probabilistic (as we will later show in 5.4.2.3).

The way in which the heteroclinic dynamics is affected by noise depends on a complex interplay between the specific system, its input, and the nature of the noise present in the system. For this reason, we cannot rely on the simple properties of the heteroclinic graph (as we did in Section 5.3 for the noiseless system) to approximate the conditional

distribution $p(X|Y)$ of the input given observed output, which we need in order to compute the Mutual Information in the system. We must instead simulate the system and collect the frequencies with which the sequences are produced. Additionally, simulating the system will not allow us to collect bi-infinite sequences of output states, so that we must settle for the collection of finite n-words instead.

Even under these constraints, studying how noise modifies the input-output Mutual Information in SSMs is a worthy endeavour, and will lead us to uncover a surprising facilitatory effect for noise on the information transmission in these systems; moreover, we will later argue that this effect is not specific to the particular system we chose to simulate here, but can be generalized to computation in SSMs in general.

In what follows, we will first present the specific system that will guide our exploration of the effects of noise on SSMs. We will then discuss how noise was implemented, and some of its important effects on the system's dynamics. Finally we will discuss the results of the simulations and their consequences.

5.4.1 The model

In order to collect the frequencies of n-words from the dynamics of a SSM when noise is present, we want a system with key desirable characteristics:

- The system is **Efficiently simulable**. That is, it should allow for the efficient simulation of its dynamics. In fact, approximating the conditional output probabilities needed to compute the $I(X;Y)$ Mutual Information in the system can quickly become computationally burdensome.

- The heteroclinic network in the system must be **small** in the number of states. The emerging network of states should be small enough that the frequency of n-words from its output sequences can be computed in a reasonable time (larger network of states is associated with larger graphs, and thus to a larger set of possible n-words).

– The heteroclinic network in the system must be **complex**. That is, the network of states should not be so small to be trivial. This would impact the generalizability of the results.

A system with these properties has been characterized in Neves and Timme (2009). To study the impact of noise on the computation performed by SSMs, we will base our analysis on the work of Neves and Timme, who consider a system of $N = 5$ delta-pulse-coupled integrate and fire oscillators with an emerging heteroclinic network of 30 states. The simulation of this system is particularly efficient because of the small number of oscillators involved, and because its dynamics can be analytically integrated between events (resets and pulses). Furthermore, the emerging network of states is small enough to allow for the approximation of the probability of output sequences within reasonable computation time, but not so small to make the endeavour trivial. We will now describe this system by specifying its form from the general blueprint presented in Equation 5.2.

The intrinsic dynamics $F(V)$ of oscillator i is defined as

$$F(V) = I - V \tag{5.6}$$

where I can be thought as a driving current to the oscillator. We further define a reset condition on the Voltage of the oscillator, such that

$$V(t) = \begin{cases} V(t) & \text{if } V(t) < \theta \\ 0 & \text{if } V(t) \geq \theta, \end{cases} \tag{5.7}$$

where θ is the firing threshold. That is, when the Voltage of the oscillator reaches the threshold, the oscillator fires (sends a pulse) and its Voltage is reset to 0.

The network of delta-pulse-coupled oscillators is connected homogeneously all-to-all with no self-connections, as in Figure 5.2.

Every time an oscillator reaches its threshold, it sends a pulse to the rest of the network. In this model, the pulse is transmitted to the other oscillators with a delay, so that the network input to the i-th oscillator for the general system in Equation 5.2 can be specified

θ	τ	ϵ	I	$V_{1,2}$	$V_{3,4}$	V_5
1	1.59646730	0.025	1.04	0	0.74039804	0.96216636

TABLE 5.1: **Set of parameters and initial conditions associated with the emergence of symmetrical periodicities** in the system of $N = 5$ oscillators described in this Section.

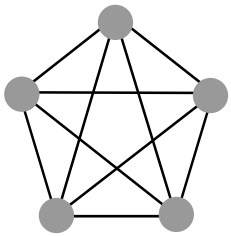


FIGURE 5.2: **Couplings between oscillators in the network.** Each oscillator is coupled to all other oscillators in the network through a connection with delay τ.

as

$$W_i(t) = \sum_{j=1, j\neq i}^{n} \epsilon\delta(t - \tau - t_j), \tag{5.8}$$

i.e. a weighted sum of incoming pulses to oscillator i from the rest of the network at time t, where pulses are received with connection delay τ, leading to instantaneous jumps in voltage of amplitude ϵ. The external input S_i to each oscillator is instead defined as

$$S_i(t) = \Delta_i \tag{5.9}$$

where $\Delta_i \ll I$ is a small detuning current to oscillator i.

Given the parameters and initial conditions reported in Table 5.1, the dynamics of the system exhibits periodic orbits characterized by the partial synchronization of its oscillators (i.e. clusters), with

$$\begin{aligned} &V_1(t) = V_2(t) \\ &V_3(t) = V_4(t) \\ &V_5(t) \neq V_4(t) \neq V_2(t), \\ &\boldsymbol{V}(t) = \boldsymbol{V}(t + kT) \text{ for } k = 1, \ldots, n \ , \end{aligned} \tag{5.10}$$

where $V_1(t)$ is the voltage of oscillator 1 at time t, $\boldsymbol{V}$ is the voltage vector, T is the period, and where $j \neq k$ for $j \neq k$ (see 5.3 for a phase depiction of a synchronized state). Note that this particular pattern of oscillator synchronization relies upon a careful initialization

of the oscillator voltages, and does not emerge otherwise. The parameters and initial conditions allowing for the emergence of this pattern have been found through a systematic numerical search (Neves, 2010).

Let us now introduce some notation to describe the clustered states in this system. We will denote the three synchronization clusters (i.e. the three groups of synchronized oscillators, $\{V_1, V_2\}$, $\{V_3, V_4\}$ and the singleton $\{V_5\}$) with the labels "a", "b" and "c". Given a vector $(1, 2, 3, 4, 5)$ in which each entry corresponds to the index of one of the 5 oscillators in the network, we can assign a cluster label to each oscillator, such that $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ a & a & b & b & c \end{pmatrix}$ is an example of clustered state. Throughout this Chapter, we will often simplify notation by reporting the cluster labels only, thus implying the associated oscillator indices. Examples of clustered states would thus be (a, b, a, c, b), or (b, c, a, a, b), or (c, b, a, b, a). Permuting the labels in one of these vectors yields a total number of possible clustered states in the system equal to $\binom{5}{2,2,1} = \frac{5!}{2!2!1!} = 30$.[1]

In what follows, we will first review and summarize key results presented in Neves (2010), applying the methods introduced in Mirollo and Strogatz (1990) to

1. present an equivalent representation for the dynamics of the discussed system of oscillators, i.e. its phase representation,
2. show the relation between the voltage and phase representations,
3. use the phase representation to derive return maps (in the form of tables of events) for the unperturbed and perturbed noiseless system and, finally,
4. characterize the heteroclinic switching of states in the system by performing a stability analysis on the return map for the perturbed system.

After having introduced these methods, we will proceed with the exposition of our results.

[1] $\binom{n}{k_1, k_2, \ldots, k_n}$ with $n = \sum_i k_i$ is the *multinomial coefficient*, expressing the number of ways in which n objects can be split in groups of respectively sizes $k_1, k_2, \ldots, k_n$.

5.4.1.1 Phase representation

The dynamics of a single uncoupled oscillator with driving current I and firing threshold θ, when considered in the parameter range $I > \theta$, is characterized by a periodic oscillation of period T_{if}, i.e. the time needed for the voltage V to increase from 0 up to the firing threshold (where it is reset), such that $V(t + kT_{\text{if}}) = V(t)$. The time evolution of the voltage of the uncoupled oscillator is defined as

$$\frac{dV}{dt} = F(V) = I - V. \tag{5.11}$$

Solving this differential equation for initial conditions $V(0) = 0$ yields

$$V(t) = I(1 - e^{-t}), \tag{5.12}$$

such that the θ firing threshold can be rewritten as $\theta = I(1 - e^{-T_{\text{if}}})$, yielding

$$T_{\text{if}} = \ln(I) - \ln(I - \theta). \tag{5.13}$$

By knowing that the voltage $V(t) = I(1 - e^{-t})$, and restricting t to the interval $[0, T_{\text{if}}]$, we can derive a formula for the time t as a function of V and I as

$$t = \ln(I) - \ln(I - V), \tag{5.14}$$

allowing us to derive the oscillator's phase $\phi(t) = \frac{t}{T_{\text{if}}}$ from its voltage as

$$\phi(t) = U\left(V(t)\right) = \frac{\ln(I) - \ln\left(I - V(t)\right)}{\ln(I) - \ln(I - \theta)} \tag{5.15}$$

such that $U : [0, \theta] \to [0, 1]$ maps the voltage representation to the phase representation of the state space of the oscillator. For $V \leq \theta$ (and thus $V < I$), U is monotonic, allowing us to also derive the inverse mapping U^{-1} from phase to voltage representation $U^{-}1\left(U\left(V\left(t\right)\right)\right) = U^{-1}\left(\phi_i\left(t\right)\right) = V(t)$. Given that $\phi_i = \frac{t}{T_{\text{if}}}$, we can straightforwardly derive the inverse of $U\left(V(t)\right)$, i.e. The function mapping the phase representation to the voltage representation. Specifically, given that $t = \phi_i(t)T_{\text{if}}$,

$$U^{-1}\left(\phi\left(t\right)\right) = V(t) = I\left(1 - e^{\phi(t)T_{\text{if}}}\right). \tag{5.16}$$

event	time	ϕ_1, ϕ_2	ϕ_3, ϕ_4	ϕ_5	event num.
$\sigma_{1,2}$	0	0	D	E	0
$\rho_{3,4}; \sigma_5$	τ'	$H_{2\epsilon}(\tau') = p_{1,1}$	$H_\epsilon(D + \tau') = p_{3,1}$	$H_{2\epsilon}(E + \tau') > 1 \to 0$	1
$\rho_{1,2}; \sigma_{3,4}$	τ	$H_\epsilon(p_{1,1} + \tau - \tau') = p_{1,2}$	$H_{2\epsilon}(p_{3,1} + \tau - \tau') > 1 \to 0$	$H_{2\epsilon}(\tau - \tau') = p_{5,2}$	2
ρ_5	$\tau' + \tau$	$H_\epsilon(p_{1,2} + \tau') = p_{1,3}$	$H_\epsilon(\tau') = p_{3,3}$	$p_{5,2} + \tau' = p_{5,3}$	3
$\sigma_{1,2}$	$\tau' + \tau + 1 - p_{1,3}$	$1 \to 0$	$p_{3,3} + 1 - p_{1,3}$	$p_{5,3} + 1 - p_{1,3}$	4

TABLE 5.2: **Table of events for the unperturbed orbit** (adapted from Neves, 2010). Here three periodic conditions are implied, i.e. $D = p_{3,3} + 1 - p_{1,3}$, $E = p_{5,3} + 1 - p_{1,3}$, and $\tau' = \frac{\tau - 1 + p_{1,3}}{2}$, which are met, e.g., for $D = .381978, E = .795680, \tau' = .119095$. Note that $H_{2\epsilon}$ denotes the reception of two pulses from an oscillator.

To characterize the relation between the phase and the voltage representations for the *coupled* oscillator, we need to show how the voltage jump caused by the reception of a pulse is mapped to a phase jump. The reception of a pulse by the coupled oscillator is associated with an instantaneous jump in voltage of amplitude ϵ. On the other hand, the amplitude of the associated phase jump ϵ_ϕ is not fixed (as the voltage function is concave down) but is rather a function g of the oscillator's phase immediately preceding the reception of the pulse at time t_ρ, i.e. $\epsilon_\phi = g\left(\lim_{t \to t_\rho^-} \phi(t)\right)$. Having shown how to map from voltage to phase and vice-versa in Equations 5.15 and 5.16, the new phase resulting from the reception of a pulse can be trivially derived as

$$H_\epsilon\left(\phi_i(t_\rho^-)\right) = U\left[U^{-1}\left(\phi_i(t_\rho^-)\right) + \epsilon\right] = \phi_i(t_\rho) \tag{5.17}$$

where $H_\epsilon\left(\phi\left(t\right)\right)$ maps a voltage jump due to the reception of a pulse at time t_ρ to the resulting phase.

5.4.1.2 Tables of events

Having specified the relation between the voltage and phase representation of the dynamics of the oscillators, we can now derive the table of events for a clustered periodic orbit in the noiseless and unperturbed system. Let us denote the production of a pulse by oscillator i with σ_i, and the reception of a pulse by oscillator j by ρ_j. The table of events for a periodic orbit in this system can then be derived as in Table 5.2. In Figure 5.3 we show the phase

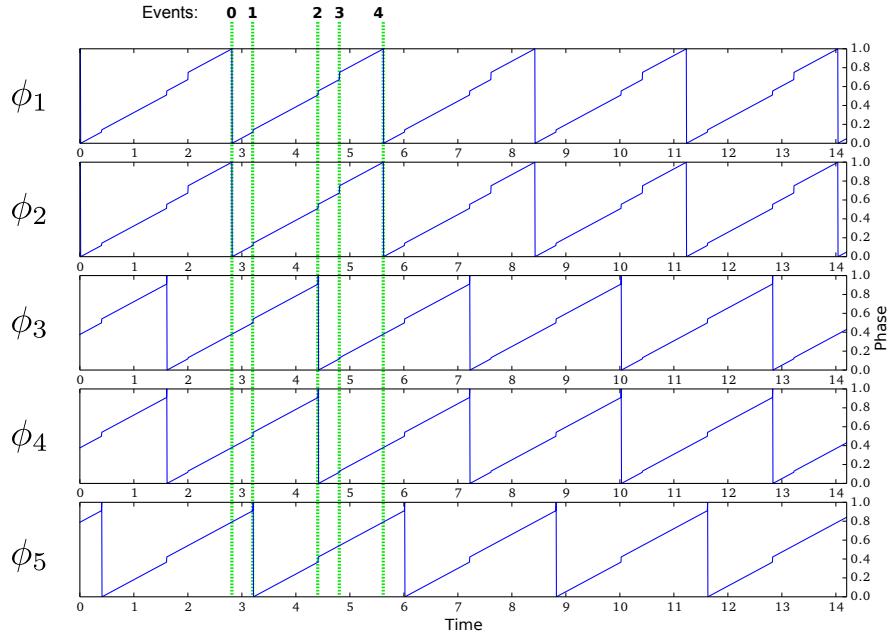


FIGURE 5.3: **Phase of a periodic orbit in the unperturbed and noiseless system of oscillators.** The dynamics of the system is characterized by discrete reset and pulse reception events. In the Figure, we highlight the events by vertical green dotted lines, and cross-reference them with Table 5.2.

of a clustered periodic orbit in the unperturbed noiseless system, cross-referenced with the events reported in Table 5.2.

After deriving the table of events for a periodic orbit in the unperturbed system, we can extend it by adding a perturbation vector to the phase of the oscillators at the reference reset, deriving a new table of events describing the phase evolution of the perturbed system between resets of the reference oscillator.

We define each component of the perturbation vector $\boldsymbol{\delta}(n)$ as the difference in phase between the oscillators in the perturbed vs unperturbed system, measured at the n-th reset of a reference oscillator, i.e.

$$\delta_i(n) := \phi_i(t_{1,n}) - \phi_i^*(t_{1,n}), \tag{5.18}$$

where $\phi_i(t_{1,n})$ denotes the phase of the perturbed system at time $t_{1,n}$ (the time of the n-th reset of oscillator 1, taken as the reference oscillator), and ϕ_i^* denotes the phase of the unperturbed system at that time. The derived table of events is shown in Table 5.3.

event	time	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5	event num.
$\sigma_{1,(2)}$	0	0	δ_2	$D+\delta_3$	$D+\delta_4$	$E+\delta_5$	0
$\rho_{3,4};\sigma_5$	τ'	$H_{2\epsilon}(\tau')=p_{1,1}$	$H_{2\epsilon}(\tau'+\delta_2)=p_{2,1}$	$H_{\epsilon}(D+\tau'+\delta_3)=p_{3,1}$	$H_{\epsilon}(D+\tau'+\delta_4)=p_{4,1}$	$H_{2\epsilon}(E+\tau'+\delta_5)>1\to 0$	1
$\rho_2;\sigma_{3,4}$	$\tau-\delta_2$	$H_{\epsilon}(p_{1,1}+\tau-\tau'-\delta_2)=p_{1,2a}$	$p_{2,1}+\tau-\tau'-\delta_2=p_{2,2a}$	$H_{\epsilon}(p_{3,1}+\tau-\tau'-\delta_2)>1\to 0$	$H_{\epsilon}(p_{4,1}+\tau-\tau'-\delta_2)>1\to 0$	$H_{\epsilon}(\tau-\tau'-\delta_2)=p_{5,2a}$	2a
ρ_1	τ	$p_{1,2a}+\delta_2=p_{1,2}$	$H_{\epsilon}(p_{2,2a}+\delta_2)=p_{2,2}$	$H_{\epsilon}(\delta_2)=p_{3,2}$	$H_{\epsilon}(\delta_2)=p_{4,2}$	$H_{\epsilon}(p_{5,2a}+\delta_2)=p_{5,2}$	2
ρ_5	$\tau+\tau'$	$H_{\epsilon}(p_{1,2}+\tau')=p_{1,3}$	$H_{\epsilon}(p_{2,2}+\tau')=p_{2,3}$	$H_{\epsilon}(p_{3,2}+\tau')=p_{3,3}$	$p_{3,3}$	$p_{5,2}+\tau'$	3
σ_2	$\tau+\tau'+1-p_{2,3}$	$p_{1,3}+1-p_{2,3}$	$1\to 0$	$p_{3,3}+1-p_{2,3}=p_{3,4a}$	$p_{3,4}$	$p_{5,2}+\tau'+1-p_{2,3}$	4a
σ_1	$\tau+\tau'+1-p_{1,3}$	$1\to 0$	$p_{2,3}-p_{1,3}$	$p_{3,3}+1-p_{1,3}$	$p_{3,3}+1-p_{1,3}$	$p_{5,2}+\tau'+1-p_{1,3}$	4

TABLE 5.3: **Table of events for the perturbed orbit** (adapted from Neves, 2010). See Figure 5.5 for a phase depiction of the events in the table, and Table 5.2 for the definition of D, E and τ'.

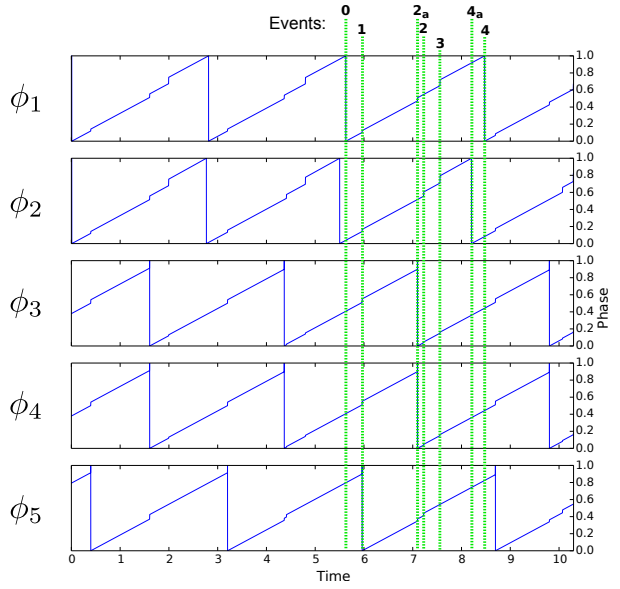


FIGURE 5.4: **Phase of system after perturbation.** The reset and pulse reception events are highlighted with green dotted lines, and cross-referenced with those in Table 5.3.

5.4.1.3 Stability analysis

It is now possible, through the return map defined by the table of events in Table 5.3 for a perturbed system in the (a, a, b, b, c) clustered state, to perform a stability analysis on the perturbation vector $\boldsymbol{\delta}(n) = \big(\delta_2(n), \delta_3(n), \delta_4(n), \delta_5(n)\big)$. [2] The stability analysis is performed by tracking the evolution of the perturbation vector from one reset of the reference oscillator to the next, which can be analytically traced thanks to the table of events for the perturbed orbit, in Table 5.3. The evolution of the perturbation vector at reset $n+1$ can then be expressed as a function of its state at reset n, that is

$$\boldsymbol{\delta}(n+1) = F\left(\boldsymbol{\delta}\,(n)\right).$$

It then becomes possible to analyze the stability of the perturbation vector by performing a linear approximation on F, such that

$$\boldsymbol{\delta}(n+1) \approx J\boldsymbol{\delta}(n).$$

[2] Note that reference oscillator 1 is here omitted, as the perturbation vector is defined in relation to its reset times $t_{1,n}$, and thus $\delta_1(n) = 0$ for all n (see Equation 5.18). That is, the return map is constructed by taking a "snapshot" of the oscillator phases every time the reference oscillator 1 fires, so that its phase at the time of the snapshot is always 0, and can thus be omitted.

where J is the Jacobian at $\boldsymbol{\delta}(n) = 0$. Assuming $\{\delta_2, \delta_5\} > 0$, and $\delta_3 < \delta_4$, the stability analysis on J reveals a single non-zero eigenvalue with its eigenvector in the direction of δ_2, and three zero eigenvalues in the direction of δ_3, δ_4 and δ_5. That is, for the system in the (a, a, b, b, c) synchronized state, only perturbations causing a positive difference in phase between oscillator 2 and reference oscillator 1 have an effect on the return map, whereas all other perturbations are cancelled within one reset of the reference oscillator. Specifically, a positive perturbation to oscillator 2 (or a negative one to oscillator 1) causes the a synchronization cluster to split, such that oscillator 2 synchronizes in time with oscillator 5, becoming the new stable cluster b, whereas oscillator 1 becomes the new singleton c, and oscillators 3 and 4 become the new unstable cluster a. That is, the new approached synchronized state has the same symmetry as the starting one. We are thus able to derive a transition rule of the form

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ a & a & b & b & c \end{pmatrix} \xrightarrow{\delta_2 > 0} \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ b & c & a & a & b \end{pmatrix} \tag{5.19}$$

where δ_2 is a difference in phase between oscillator 2 and reference oscillator 1 caused by a positive perturbation to oscillator 2, or a negative one to oscillator 1. Because of the symmetry of the system, a second transition is possible from the (a, a, b, b, c) clustered state, which is easily derived by considering that the perturbation analysis results hold unchanged when using oscillator 2 as a reference, yielding

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ a & a & b & b & c \end{pmatrix} \xrightarrow{\delta_1 > 0} \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ c & b & a & a & b \end{pmatrix}, \tag{5.20}$$

where δ_1 is a difference in phase between oscillator 1 and reference oscillator 2 caused by a positive perturbation to oscillator 1, or a negative one to oscillator 2.

A consequence of the two transition rules in Equations 5.19 and 5.20 is that, starting from the (a, a, b, b, c) clustered state, which of the two clustered states (b, c, a, a, b) or (c, b, a, a, b) is approached after a perturbation to the unstable cluster is determined by which of the two oscillators in the cluster receives the strongest perturbation. In Figure 5.5, we perturb one of the oscillators in the unstable cluster and show the effect of the perturbation on the reset timings of the oscillators in the system, highlighting the different synchronized clusters in different colors.

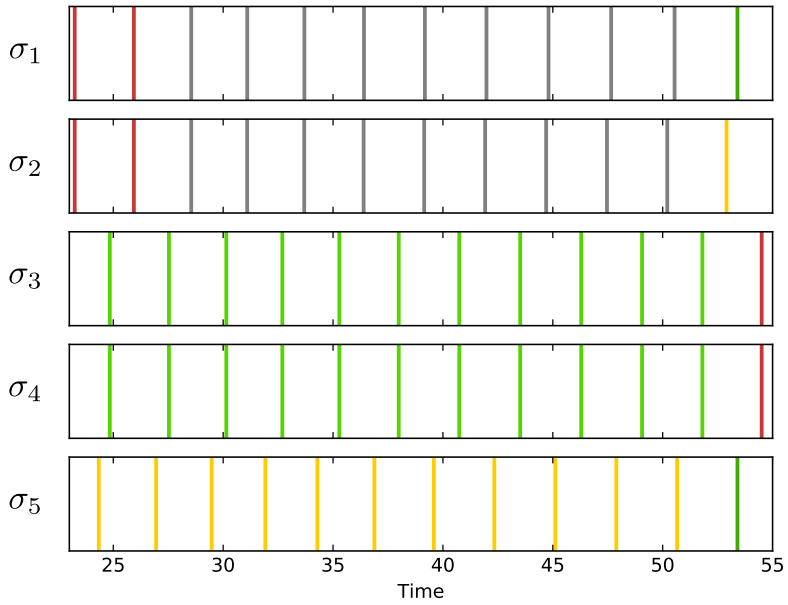


FIGURE 5.5: **Reset timings of oscillators in the noiseless system for a positive perturbation applied to oscillator 2 (at t ≈ 28).** The stable cluster is highlighted in green, the unstable cluster in red and the singleton in yellow. In grey, we highlight the de-synchronized unstable cluster. Note how, after a transient de-synchronization, the oscillators in the system re-synchronize in a new clustered state, with the same symmetry as the initial clustered state. In the approached state, the original stable cluster becomes the new unstable cluster, the perturbed oscillator of the original unstable cluster becomes the new singleton, while the other synchronizes with the old singleton to form the new stable cluster.

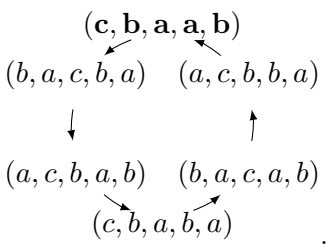
The heteroclinic graph defined in 5.1 for this system can be derived by defining the set of vertices $V(G)$ to be equal to the set of clustered states in the system, and the set of edges $E(G)$ to be equal to the set of transitions between states, which can be obtained by permutation of the indices of the oscillators in the transition rule in Equation 5.19, i.e.

$$E(G) = \left\{ \begin{pmatrix} r(\ 1\ 2\ 3\ 4\ 5\ ) \\ \downarrow\downarrow\downarrow\downarrow\downarrow \\ a\ a\ b\ b\ c \end{pmatrix} \xrightarrow{\delta_{r(2)}>0} \begin{pmatrix} r(\ 1\ 2\ 3\ 4\ 5\ ) \\ \downarrow\downarrow\downarrow\downarrow\downarrow \\ b\ c\ a\ a\ b \end{pmatrix} \quad \middle| \quad \text{for all } r \in \mathcal{P}(1,2,3,4,5) \right\} \tag{5.21}$$

where $\mathcal{P}(1,2,3,4,5)$ is the set of all possible bijections from the set of oscillator indices to itself (i.e. all possible permutations of the oscillator indices). The set of edges $E(G)$ could have equivalently been obtained by permutation of the oscillator indices in the second transition rule defined in Equation 5.20, as each of the two transition rules can be obtained by permutation of the oscillator indices in the other.

When perturbed through the application of a fixed vector of detuning currents $\mathbf{\Delta} =$

$(\Delta_1, \ldots, \Delta_5)$ to its oscillators (as defined in Equation 5.9), the system performs a 3-winners-take-all computation separating the three highest detuning currents from the lowest two. For example, given $\Delta_1 > \Delta_2 > \Delta_3 > \Delta_4 > \Delta_5$ and an initial clustered state (c, b, a, a, b), the system traces a periodic sequence of states of the form



Note that, by the transition rules in Equations 5.19 and 5.20, observing this orbit implies that $\{\Delta_1, \Delta_2, \Delta_3\}$ are all larger than $\{\Delta_4, \Delta_5\}$.

In the absence of noise, we can compute a baseline Mutual Information for this system by the results discussed in 5.3. Specifically, given 5.5, the Mutual Information $I(X;Y)$ between the input set X of all possible orderings for the currents in the $\mathbf{\Delta}$ vector and the output set Y of all possible periodic sequences of states can be derived as $I(X;Y) = \log_2 \binom{5}{3} = \log_2 10 \approx 3.32$.

5.4.2 Effects of noise on switching dynamics

In the previous Sections, we have characterized the dynamics of our small system of oscillators in the absence of noise. We are now ready to investigate how noise modifies these dynamics.

We complete the specification of the general system in Equation 5.2 by adding a Gaussian noise source η_i to each of the oscillators in the network, implemented in the form of an external random pulse generator. We do so in order to preserve the between-event integrability of the system, keeping its simulation efficient. The cumulative current produced by the pulses from the noise source in unit time approximates a Gaussian distribution with mean 0 and variance a^2. Specifically, each oscillator is connected to two Poisson pulse generators, producing pulses with rate $\frac{\lambda}{2}$ and amplitude respectively of a and $-a$ (i.e. one

generates excitatory input, the other inhibitory). In a unit of time, each pulse generator produces a Poisson-distributed number of pulses, which we will denote as respectively K^+ and K^-, and a cumulative current of respectively $C^+ = aK^+$ and $C^- = aK^-$. The mean E and variance Var of C^+ and C^- can be derived as follows

$$\begin{aligned}
E(C^+) = E(\ \ aK^+) &= a\ E(K^+) &&= a\ \frac{\lambda}{2}, \\
E(C^-) = E(-aK^-) &= -a\ E(K^-) &&= -a\ \frac{\lambda}{2}, \\
\mathrm{Var}(C^+) = \mathrm{Var}(\ \ aK^+) &= a^2\mathrm{Var}(K^-) &&= a^2\frac{\lambda}{2}, \\
\mathrm{Var}(C^-) = \mathrm{Var}(-aK^+) &= a^2\mathrm{Var}(K^-) &&= a^2\frac{\lambda}{2}.
\end{aligned}$$

Together, in unit time the two generators produce an overall current of $Z = C^+ + C^-$. Given the independence of the the two C^+ and C^- random variables, we can derive the mean and variance of Z as follows

$$E(Z) = E(C^+ + C^-) = E(C^+) + E(C^-) = 0, \tag{5.22}$$

$$\mathrm{Var}(Z) = \mathrm{Var}(C^+ + C^-) = \mathrm{Var}(C^+) + \mathrm{Var}(C^-) = a^2\lambda. \tag{5.23}$$

For sufficiently high λ rate and small amplitude a, the pulses produced by the two generators approximate the application of a continuous current. For this reason, the variance of the cumulative current produced in unit time $\mathrm{Var}(Z)$ becomes a control parameter for the noise "strength", thus governing its effects on the system.

In Figure 5.6 we show the results of a simulation where we measure the timings of resets for a decoupled oscillator with dynamics $\frac{dV}{dt} = I - V + \eta$, where the η noise source is defined as discussed in the previous paragraphs, and where we vary the λ and a noise parameters, measuring the reset timings variance for a number of reset equal to $100,000$. In the Figure, it is possible to observe that varying a or λ has approximately the same effect on the reset timings as long as the noise variance $\mathrm{Var}(Z)$ is kept the same. To manipulate the noise variance in our simulations, we will keep the λ pulse rate fixed, and act on the squared amplitude a^2 instead. This is because increasing λ also linearly increases the computation time for the event-based simulations, as more pulses are produced by the pulse generators. In contrast, manipulating a^2 does not lead to an increase in simulation time.

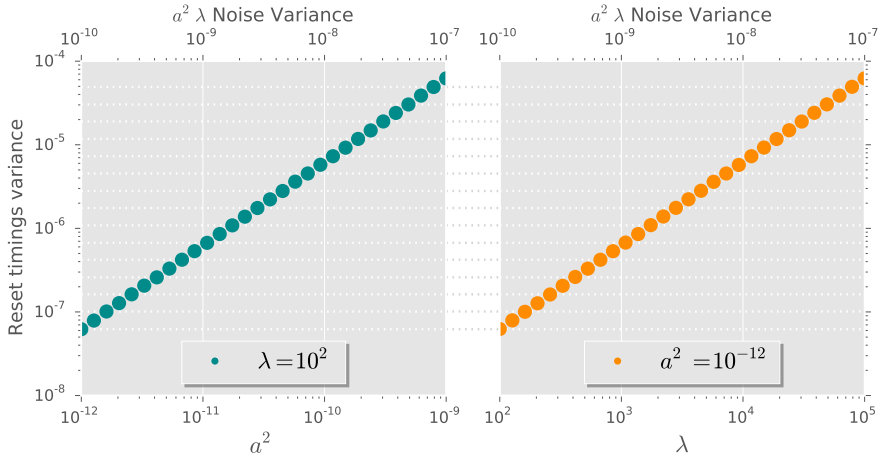


FIGURE 5.6: **Reset timings variance for a range of noise parameter values.** On the left, a noise rate $\lambda = 10^2$ is fixed, and the squared noise amplitude a^2 is manipulated. On the right, a squared noise amplitude $a^2 = 10^{-12}$ is fixed, and the noise rate λ is manipulated. Note how, for the same resulting noise variance in the manipulation of the two parameters (shown in the top axis), the resulting reset timings variance in the left and right plot is approximately the same (highlighted by the dotted horizontal lines).

5.4.2.1 Switching time for noiseless system

As the effect of perturbations to the stable clusters are cancelled by resets elicited from the reception of pulses in the corresponding oscillators (i.e. events 2 and 4 in Table 5.2), the switching time between states is driven primarily by the magnitude of the difference $D = \Delta_1 - \Delta_2$ between the currents Δ_1 and Δ_2 to the two oscillators in the unstable cluster. In fact, when the difference between detuning currents is zero i.e. when the two oscillators receive the same current, they stay synchronized and thus no transition is induced; when $D \neq 0$, instead, stronger differences $|D|$ are associated with a faster de-synchronization, and thus a faster switching to the next state in the heteroclinic network. We show this effect in Figure 5.7.

Note that the discrete jumps in switching time present in Figure 5.7 are due to the fact that smoothly increasing/decreasing the magnitude $|D|$ of differences in currents to the unstable cluster eventually leads to the addition/subtraction of a reset in the transition between clustered states, thus suddenly causing a jump in switching time. Furthermore, note that the switching time increases exponentially as the magnitude decreases. In Figure

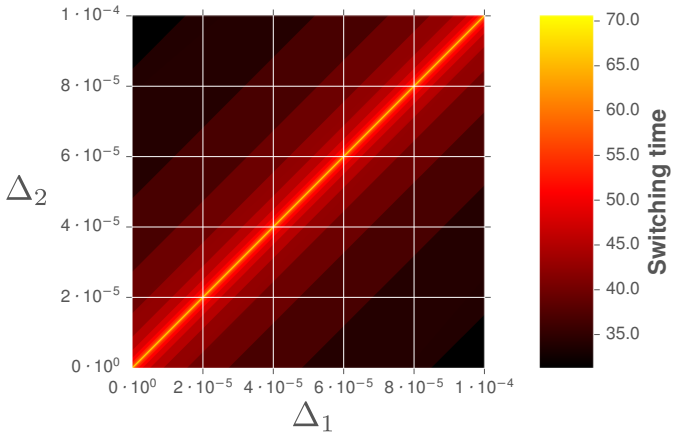


FIGURE 5.7: **Switching time as a function of currents applied to oscillators in unstable cluster.** The Figure shows how the switching time depends primarily on the difference between the input currents to the oscillators in the unstable cluster, rather than their magnitude.

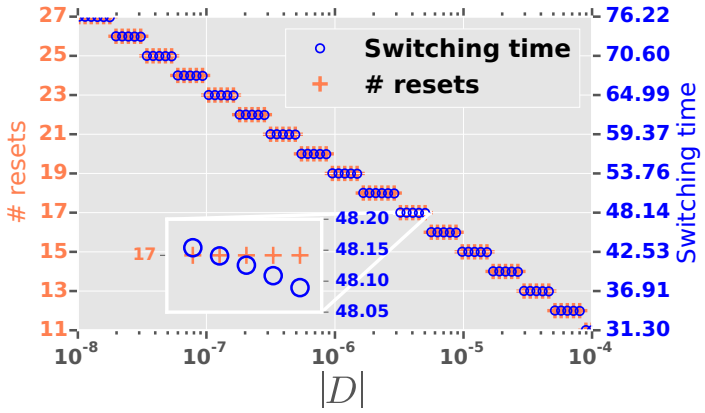


FIGURE 5.8: **Switching time and number of resets as functions of D difference.** This Figure highlights the exponential relationship between the difference in currents to the unstable cluster ($D = \delta_i - \delta_j, \delta_i > \delta_j$) and the switching time. In particular, notice how the discrete time jumps are related to the underlying change of the number of resets in switching.

5.8 we highlight both the addition/subtraction of resets for varying magnitude $|D|$ of the difference, and the exponential relationship between the latter and switching time.

5.4.2.2 Switching time for system with noise

In Figure 5.9 we show the mean switching time and its standard deviation for the unperturbed noisy system, computed from runs of $k = 1000$ transitions, for a range of 20 values of the noise Variance $\mathrm{Var}(Z)$ sampled logarithmically over the $[10^{-27}, 10^{-7}]$ interval. As

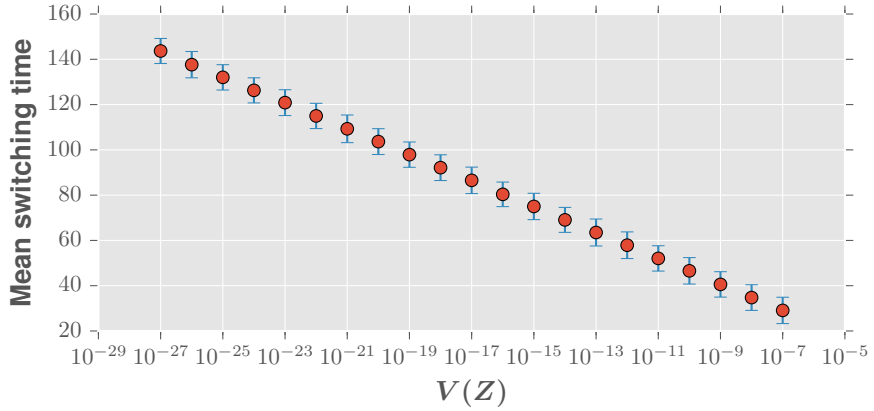


FIGURE 5.9: **Mean and standard deviation of switching time for varying noise variance** $\mathrm{Var}(Z)$**.** We denote one standard deviation above and below the mean by a blue bar. In the absence of input, the mean switching time in the system increases exponentially as the noise strength decreases.

the Figure shows, greater noise variance (i.e. greater noise strength) causes the switching to accelerate in a predictable fashion. This is because (as shown in Neves, 2010) stronger noise causes the dynamics of the system to spend less time on average near the attractors, and thus to not suffer from their associated exponential slowdown.

5.4.2.3 Errors in state switching

It is possible to look at each state-to-state transition in the heteroclinic network as a computation on the input currents, where the result of the comparison between the two input currents to the unstable cluster determines the next state (between two possible) in the sequence of approached states. In this sense, noise can lead to "errors" in switching when it forces the system to approach the wrong state, i.e. the state associated with the wrong result when looking at the comparison between input currents.

To study how the relative strength between input and noise affects the frequency of errors in switching, we simulate the system by fixing the noise variance to $\mathrm{Var}(Z) = 10^{-9}$ and the vector of detuning currents to 0 with the exception of one component (Δ_1), which we vary to control the Signal-to-Noise Ratio (SNR), defined here as $\frac{(\Delta_1)^2}{\mathrm{Var}(Z)}$. For each level of SNR, we record 100 transitions from an initial state, and compute the frequency with which the right vs wrong state is approached after de-synchronization due to the Δ_1 current to the

unstable cluster. We report the results in Figure 5.10, where we show that the probability for the system to perform the correct transition (and thus to correctly compare the two input currents) drops to chance for low SNR levels.

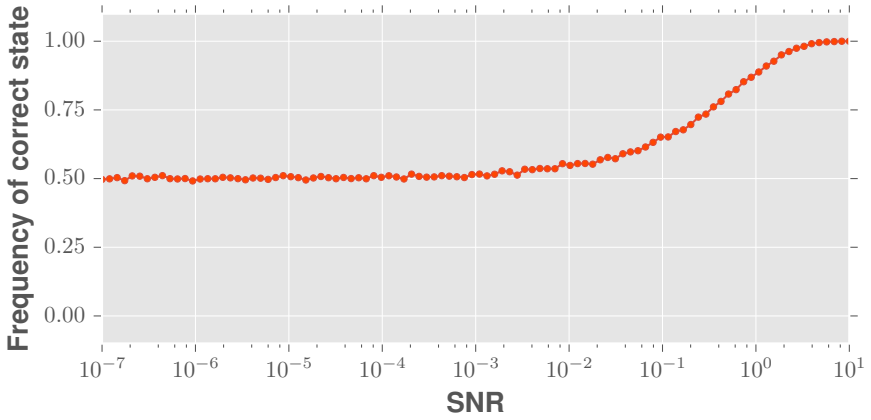


FIGURE 5.10: **Frequency with which the correct state is approached for a range of SNR levels**. We simulate a transition in the system by initializing the system to be on an initial clustered state $s_0 = (a, a, b, b, c)$, fixing the noise variance $\mathrm{Var}(Z)$ to a value of 10^{-9}, and varying the magnitude Δ of the input vector of currents, with $\Delta = (\Delta_1, 0, 0, 0, 0), \Delta_1 > 0$. For each level of SNR, we collect the number of times n_1 and n_2 each of the two possible ending states $s_1 = (c, b, a, a, b)$ and $s_2 = (b, c, a, a, b)$ is approached over $n_{\mathrm{tot}} = 100$ trials. Finally, we obtain the frequency of each of the two states as $\frac{n_1}{n_{\mathrm{tot}}}$ and $\frac{n_2}{n_{\mathrm{tot}}}$. In the noiseless system, the transition rule prescribes s_1 as the correct state to be approached given s_0 as the initial state and $(\Delta_1, 0, 0, 0, 0)$ with $\Delta_1 > 0$ as input; in the Figure, we show the frequency $\frac{n_1}{n_{\mathrm{tot}}}$ with which state s_1 is approached, for each level of SNR.

5.4.3 Mutual Information in noisy system

In the previous Sections, we characterized the effect of noise on the local switching dynamics, showing that noise accelerates the overall switching rate and that it causes errors in the transition between states (if we see transitions as computations on the input, where the input currents to the oscillators of the unstable cluster are compared).

In the next Sections, we will show that, globally, these two effects of noise can lead to an increased capacity and speed of information transmission in the system (seen as a re-encoder of its input currents into output sequences). We do so by computing the Mutual Information $I(X;Y)$ and its rate in this system for varying noise strength.

We will now describe the input and output sets used to test the system in the following simulations, and introduce a Signal-to-Noise Ratio (SNR) to quantify the noise strength compared to the input.

Input set. We first define the X input set as discussed in Section 5.3 for the noiseless case, i.e. as the set of all possible orderings of a vector of input currents. As shown in 5.4.2, the magnitude of the difference between the currents has a key role in determining the robustness to noise of the system (all things equal, greater differences lead to less errors in switching). For this reason, we will simulate the system for a range of vectors of detuning currents, differing for the specific way in which the differences between the currents are chosen. Specifically, a generic vector $\mathbf{\Delta}^g$ of detuning currents (which we call *generating vector* as it generates the X input set by permutation of its entries), is defined as

$$\Delta_i^g = \begin{cases} b & i = 1 \\ \Delta_{i-1} + D_{i-1} & i = 2, \dots, N, \end{cases} \tag{5.24}$$

where $b \in \mathbb{R}$ is some constant, and $D_1, D_2, \dots, D_{N-1}$ are independent and identically distributed random variables from some chosen probability distribution F. That is, the differences between consecutive currents in the $\mathbf{\Delta}^g$ vector are chosen randomly from some distribution F, which we will vary in order to test the system for a range of conditions.

Output set. In the noiseless case we could define the output set Y as the set of all possible periodic sequences given by the persistent cyclic switching between states due to a fixed vector of input currents to the oscillators. On the other hand, we have shown in Section 5.4.2.3 that noise disrupts the cyclic switching by sometimes causing the wrong state to be approached. For this reason, whereas before we could consider a finite set of periodic infinite sequences, we must now choose a finite set of finite sequences as an output set, as our results will be derived through simulation.

A natural choice of output set given these conditions is the set of all possible n-words in the vertex shift $\mathbf{X}_G$ defined by the heteroclinic graph G (see Equation 5.1). That

is, the set of all possible sequences of n states that the heteroclinic dynamics of the system can trace in the network of states. In the following Sections, we will set $n = 11$ as a reasonable trade-off between the increase in computation time for the analysis of the data and the clarity of exposition of our results. Note that each n-word (each sequence of n states) in the output set is now effectively considered a single output symbol.

Signal-to-Noise Ratio. Given the probability distribution F from which the difference between consecutive currents are generated (as specified in Equation 5.24), we define the SNR as

$$\text{SNR} = \frac{E(D^2)}{\text{Var}(Z)}, \tag{5.25}$$

where D is a random variable distributed as F, $E(D^2)$ is the expected value of its square, and $\text{Var}(Z)$ is the variance of the noise random variable Z (i.e. λa^2, as shown in Section 5.4.2). Note that we chose to use the D difference random variable as a measure of input strength because, as we discussed in Sections 5.4.2.2 and 5.4.2.3, a stronger difference between currents to the unstable cluster is associated with a switching which is faster and more robust against noise.

For each input set tested in the simulations presented in the following Sections, we measure the Mutual Information of the system for 20 levels of SNR (sampled logarithmically from the $[\frac{1}{10}, 10]$ interval), and testing each of the 30 states in the network as initial conditions. For each run (i.e. for each level of SNR and initial condition), we record a long-running switching sequence of length $k = 1000$, from which we collect the needed output n-words through the n-th higher word code in Equation 3.1. In this way, for each level of SNR we are able to collect 29700 n-words, i.e. the system's output sequences.

We test the system for three distributions F of the differences between consecutive detuning currents.

In a first set of simulations, we generate the input set X by permutation of a vector of currents $\mathbf{\Delta}^g$ where the difference between consecutive currents is fixed to some value d. When considering the generic $\mathbf{\Delta}^g$ vector introduced in Equation 5.24, this is equivalent

to defining the distribution F of the D_i differences to be the deterministic distribution localized at d. The results are reported in panel A of Figure 5.11, where we observe an increase of up to 14% in the Mutual Information and one of up to 12% in the Mutual Information Rate, with respect to the baseline Mutual Information and Mutual Information Rate for the noiseless system.

In a second set of simulations, we test 100 different input sets, each from a vector of currents in which the differences D_i between consecutive currents are generated from the $U(0, 10^{-5})$ uniform distribution on the $(0, 10^{-5})$ interval. The squared expectation $E(D^2), D \sim U(a, b)$ needed to specify how the SNR is computed can be derived as

$$\begin{aligned} E(D^2) &= \int_a^b x^2 \cdot \frac{1}{b-a} dx \\ &= \frac{1}{b-a} \int_a^b x^2 dx \\ &= \frac{1}{b-a} \frac{b^3 - a^3}{3} \\ &= \frac{a^2 + ab + b^2}{3}. \end{aligned}$$

Like for the first set of simulations, the results (shown in panel B of Figure 5.11) suggest an increase in Mutual Information for intermediate levels of noise.

In a last set of simulations (panel C of Figure 5.11), we test 100 input sets from input vectors with differences D_i generated from a distribution $10^{N(\mu,\sigma)}$ where $N(\mu, \sigma)$ is the normal distribution with mean μ and variance σ. The reason for this rather exotic distribution is that, as we have seen in Section 5.4.2, the relationship between the difference in Δ currents and the switching time is exponential (Section 5.4.2.1), as is the relationship between noise strength and switching time (Section 5.4.2.2). We thus wanted the differences between currents to be random but within approximately the same order of magnitude (which we fixed to 10^{-6}, yielding $\mu = -6$), with some controlled deviation (the σ parameter, which we vary in simulation). The squared expectation $E(D^2), D \sim 10^{N(-6,\sigma)}$ needed to specify

how the SNR is computed can be derived as

$$
\begin{aligned}
E(D^2) &= \int_{-\infty}^{\infty} 10^{2x} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \\
&= \int_{-\infty}^{\infty} 10^{2x} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \frac{1}{\sigma\sqrt{2\pi}} dx && \text{but } 10^{2x} = e^{\ln(10)2x}\text{, so that} \\
&= \int_{-\infty}^{\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2} + \ln(10)2x} \frac{1}{\sigma\sqrt{2\pi}} dx \\
&= \int_{-\infty}^{\infty} e^{-\frac{x^2 - 2xk + \mu^2 + k^2 - k^2}{2\sigma^2}} \frac{1}{\sigma\sqrt{2\pi}} dx && \text{where } k = (\mu + 2\sigma^2 \ln(10)) \\
&= \int_{-\infty}^{\infty} e^{-\frac{(x-k)^2 + \mu^2 - k^2}{2\sigma^2}} \frac{1}{\sigma\sqrt{2\pi}} dx \\
&= e^{\frac{k^2 - \mu^2}{2\sigma^2}} \int_{-\infty}^{\infty} e^{-\frac{(x-k)^2}{2\sigma^2}} \frac{1}{\sigma\sqrt{2\pi}} dx && \text{Using that integral of Gaussian is equal to 1} \\
&= e^{\frac{(\mu + 2\sigma^2 \ln(10))^2 - \mu^2}{2\sigma^2}} \\
&= e^{\frac{4\sigma^4 \ln(10)^2 + 4\mu\sigma^2 \ln(10)}{2\sigma^2}} \\
&= e^{2(\sigma^2 \ln(10)^2 + \mu \ln(10))}.
\end{aligned}
$$

The results from this last set of simulations also show an increase in Mutual Information and its rate for a range of SNR levels.

How to explain the noise-dependent facilitation for the transmission of information in this system?

At the beginning of this Chapter, we have discussed how, in the absence of noise, the perturbation-driven transition between saddle states in a heteroclinic network is determined by the largest perturbation component in the direction of the unstable manifolds of the saddle which the system has approached. In this way, a noiseless heteroclinic network, at each saddle state, performs a "comparison" between the perturbation components, determining which saddle state is approached next. For systems with complex heteroclinic networks, a fixed vector of detuning currents can induce a persistent switching between states, realizing heteroclinic cycles.

In Section 5.4.2.3, we have shown that, when noise is present, the switching becomes probabilistic; noise causes errors in switching, where the probability of performing the "correct" transition approaches chance as noise increases. Moreover, we have shown that noise accelerates switching (see Section 5.4.2.2). What our simulations reflect is the existence of

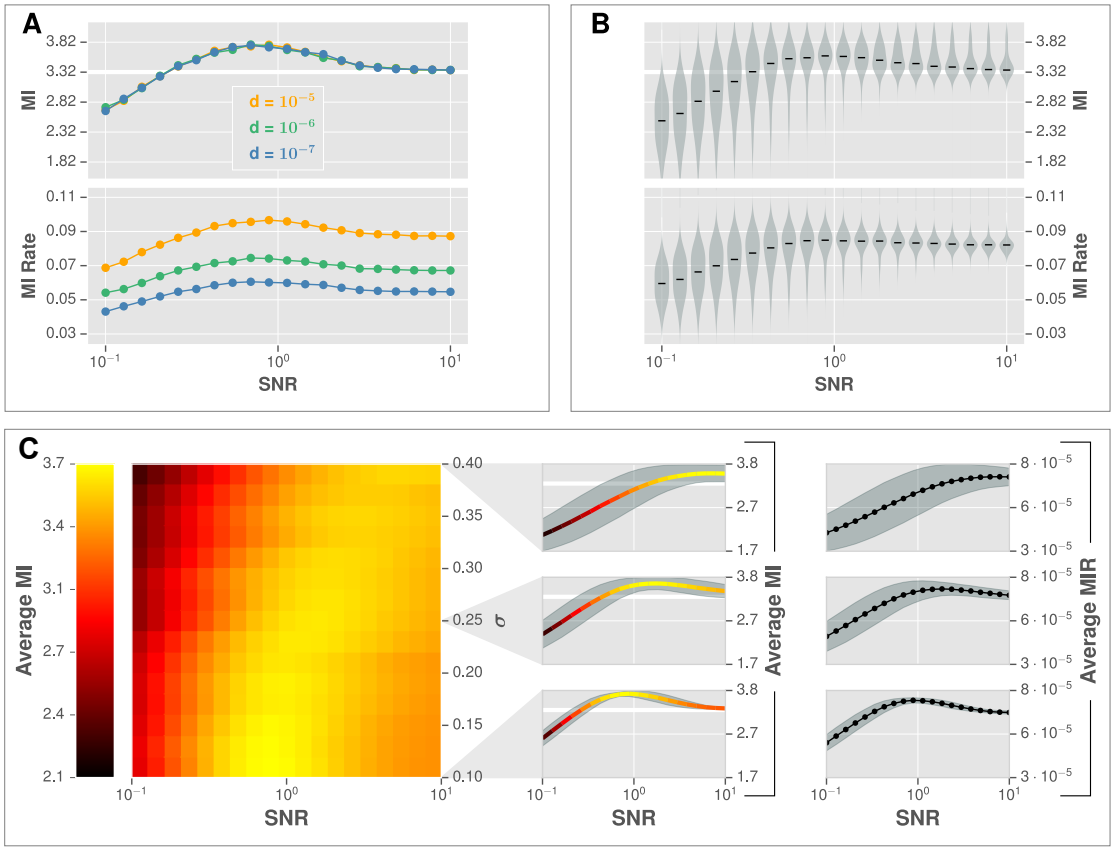


FIGURE 5.11: **Simulation results for system of N = 5 oscillator with input from different distributions.** (A) Mutual Information (MI) and its rate (MIR) for three input sets each generated from an ordered vector of equispaced currents, with the difference between one current and the next determined by a parameter d. (B) Distribution (in grey) and median (in black) of 100 values of MI and MIR computed for each SNR level from randomized input sets. Each input set is generated from an ordered vector of currents where the difference between each current and the next is randomly extracted according to a uniform distribution on the $(0, 10^{-5})$ interval. The MI and MIR distributions, shown rotated and mirrored for each SNR level, are estimated through Gaussian kernel density estimation. (C) To the right, a heat-map visualization of the average MI for 100 generating vectors with differences distributed as $10^{N(-6,\sigma)}$, where $N(-6, \sigma)$ is a Gaussian distribution with mean -6 and variance σ. We test 20 levels of σ. In the middle, selected slices for three values of σ. To the right, the averaged MIR. In the MI plots of each panel, a thick white line denotes the baseline MI value as computed for the noiseless system (see Section 5.3).

a "sweet spot" in the noise levels, where noise causes occasional errors in switching, allowing the system to explore more of the underlying network of states and thus perform comparisons that are not accessible in the noiseless case (as the system is stuck in a cycle, performing the same comparisons over and over again). If the noise is too high, then the exploration advantage is nullified by the unreliability of the switching: the output sequences become more and more independent from the input. If it is too low, then the

result of each comparison (i.e. each transition) is reliable, but only a limited number of comparisons is ever made. Between these extremes, however, there is a region of noise levels where the trade-off between exploration of the network of saddle states and reliability of the comparisons performed at each state is favourable, such that predictable orbits still exist (see the audio-visual demonstration in Carmantini, 2016), and leading to an increase in the Mutual Information between the system's input and output. Additionally, as discussed in Section 5.4.2.2, noise accelerates switching, explaining the increase in the Mutual Information Rate of the system for intermediate levels of noise. In Figure 5.12 we present the frequency for which each state is visited and the frequency of error for each state, for low/intermediate/high levels of noise in runs of $n = 1000$ recorded states.

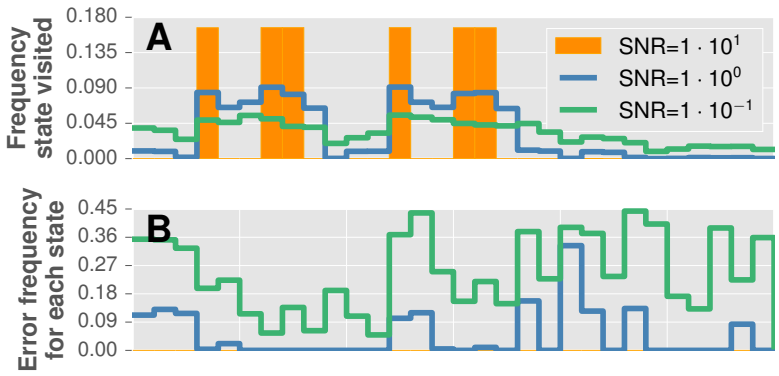


FIGURE 5.12: **Frequency with which each of the 30 states in the system is visited and frequency of errors in switching at each state for a recorded sequence of k = 1000 states and three levels of SNR.** For high SNR, only six of the states are ever visited, as the system's switching is stuck in a cycle. For intermediate SNR, errors in switching allow the system to explore more of the network of states, thus performing a larger number of computations on the input. For low SNR, a large number of errors in switching renders the computations very unreliable, thus overtaking the advantage given by the increased exploration of the network of states.

5.5 Conclusions

By analyzing how the output sequences produced by the dynamic state-switching of a simple State Switching Machine depend on the strength of its input and noise, we unveiled a facilitatory effect of noise in the computation performed in such systems. In particular, by measuring the Mutual Information and its Rate in the simulated system, we show the presence of a trade-off between increased exploration of the network of states and the

growing unpredictability in the heteroclinic switching due to increasing levels of noise. The discussed trade-off does not depend on the specific characteristics of the simulated SSM, but only on the saddle instability and presence of a network of states. For this reason, the noise-dependent facilitatory effects we have unveiled are general and hold for any realization of SSM. This is especially relevant when considering the possibility of physical implementations of SSMs. In fact, "imprecise" but faster and energy efficient microprocessors are being explored in industry (Chakrapani et al., 2007, Palem, 2005). If SSMs can be realized in physical systems, they could give rise to processors that function at a controlled continuum of regimes, from a slower/precise regime, to a faster/imprecise one.

SSMs can support Finite-State computations. In particular, a constructive mapping between arbitrary graphs and heteroclinic networks has been described in Ashwin and Postlethwaite (2013), which implies that SSMs are in fact equivalent in power to FSMs, and can thus implement *any* Finite-State computation. For this reason, the heteroclinic dynamic framework is a promising one in the search of symbolic computation in real neural systems. As a matter of fact, heteroclinic computation is hypothesized to play a role in the computation performed in the olfactory systems of some animals (Afraimovich et al., 2004, Ashwin and Marc, 2005b, Huerta et al., 2004, Laurent et al., 2001). [3]

While this framework allows us to more closely approach an appropriate level of description for symbolic computation in real neural systems (when compared with the previously discussed results on RNNs) , it also suffers from some key disadvantages. In particular, systems implementing the heteroclinic switching of states are exceptionally susceptible to perturbations. That is, persistent heteroclinic switching can only be maintained when perturbations due to noise or input have extremely small magnitude. If the level of perturbation is too high, the dynamics becomes disrupted and the switching ceases. The presence of this extreme sensitivity is not particularly plausible in real neural systems, where a reasonable degree of robustness to perturbation is to be expected. Secondly, in the unperturbed

[3]Importantly, our results show that these systems could actively harvest noise to facilitate their information processing capabilities, contributing to similar considerations from the field of "stochastic facilitation" (Magalhães and Kohn, 2011, McDonnell and Ward, 2011, Wiesenfeld et al., 1995, Zeng et al., 2000), which more generally explores and characterizes how neural systems can benefit from the presence of noise.

heteroclinic system the switching between one state and the next is asymptotic, where the final state is only reached at the limit as time goes to infinity. It is doubtful whether this appropriately reflects some characteristics of real dynamics in neural populations.

The limits of heteroclinic dynamics push us in the next Chapter to explore an alternative dynamical framework, which promises to deliver robust state switching in a more neurally plausible way.

Chapter 6

Switching in slow-fast dynamical systems

In Chapter 4 of this thesis, we introduced several objects which we used to derive a constructive mapping between a range of models of computation and the dynamics of Recurrent Neural Networks. While the architecture we put forward has a number of desirable characteristics, such as its modularity and the transparency of its operation in relation to the simulated models, its biological relevance is limited. Together with more general concerns about the biological relevance of RNNs, the fractal encoding at the core of the definition of our architecture makes its dynamics exceptionally susceptible to noise. In order to escape the shortcomings of our proposed architecture, we turned in Chapter 5 to the study of a class of neural models of stronger biological plausibility, which we named State-Switching Machines. SSMs can support Finite State Machine computation, and a constructive mapping between FSMs and SSMs has been defined in previous work (Ashwin and Postlethwaite, 2013). These characteristics make SSMs a good candidate for the search of a more general constructive mapping between the symbolic dynamics of formal systems and neuronal dynamics. As the results in the previous Chapter show, it is important to understand how noise can modify the computational capabilities of the neural systems we analyze. In Chapter 5 we do just that, showing that noise in systems supporting the

heteroclinic switching of states (SSMs) is not necessarily disruptive, but can actually facilitate the transmission of information in these systems. Nevertheless, heteroclinic dynamics is still only possible when perturbations due to noise or input are vanishingly small in magnitude, which is not particularly plausible from a biological standpoint. The asymptotic nature of the state switching in this class of systems, which we discussed in the previous Chapter, is also a point of concern.

In this Chapter we further our search for biologically relevant models of neural dynamics which can support symbolic computation, by showing evidence that an important class of smooth excitable circuit models, i.e. *slow-fast* neural models, can realize Finite-State computation through a controlled switching between clustered states, the same mechanism underlying Finite-State computation in the heteroclinic computing framework discussed in Chapter 5. Importantly, the same constructive mapping between arbitrary graphs and heteroclinic networks described in Ashwin and Postlethwaite (2013) could lead in future work to similar advancements in the slow-fast framework we introduce.

The defining characteristic of a slow-fast dynamical system is the presence of dynamical variables that evolve on different timescales; some of the variables only change slowly over time, whereas other can change very quickly. A slow-fast system is usually expressed as

$$\begin{aligned} \epsilon\dot{x} &= f(x,y) \\ \dot{y} &= g(x,y) \end{aligned} \tag{6.1}$$

where $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ are known as respectively the *fast* and *slow* variables, and where $0 < \epsilon \ll 1$.

By using slow-fast systems, we can reproduce the delayed pulse-coupling underlying the heteroclinic switching dynamics in systems such as the one described in Neves and Timme (2009). In fact, through the phenomenon of *delayed* or *dynamic bifurcations* (Benoît, 1991), we can design slow-fast systems in which positive pulse-like inputs to a slow variable causes the production of a "spike" through the fast variable after a time delay τ, which can be computed analytically via a *way-in, way-out* function. This is done by endowing the slow-fast system with an *activity-induced transcritical canard*, introduced in Rodrigues

et al. (2016). We can then couple multiple slow-fast nodes in a network to study the emergence of clustered states of synchronizations, and eventual switching (yet, not heteroclinic) dynamics between these states, as we did in Chapter 5 for the $N = 5$ system of Leaky Integrate-and-Fire oscillators.

The goal of this endeavour is to obtain a system evolving on a continuous phase space (rather than a space with discontinuities as in the delayed pulse-coupled system described in Chapter 5). By doing so, we gain access to the possibility of using numerical continuation techniques to search for stable and unstable periodicities (clustered states) in the phase space, and the bifurcation points that connect them in parameter space. This would allow, for example, to understand how the number of periodic states in the network depends on a given parameter. Additionally, slow-fast systems can be made to be very robust to noise and perturbations. While 2D slow-fast systems such as the one we will introduce in what follows are only partially robust, it has been shown that 3D slow-fast systems can support robust dynamics (Desroches et al., 2012). In future work, we will extend the basic model in this Chapter in order to obtain a robust input-driven state switching.

In the following Sections, we show preliminary results on the emergence of clustered states in a network of slow-fast nodes, akin to what shown in Chapter 5.

6.1 The model

The model we consider is a network of oscillators with topology as in 5.2, where each node i is a bi-dimensional slow-fast system (inspired by Rodrigues et al., 2016) defined by the equations

$$\dot{p}_{i1} = \epsilon \left((p_{i2} - q_{i1})(\alpha - p_{i2}) + \sum_{i \neq j}^{N} g p_{j2} \right) + I_i(t) \tag{6.2}$$

$$\dot{p}_{i2} = p_{i2}(p_{i1} - q_{i2}) \tag{6.3}$$

where $\epsilon > 0$ is a small parameter, $\sum_{i \neq j}^{N} g p_{j2}$ is input from other oscillators with coupling strength g, $I_i(t)$ is external input to oscillator i, and

$$q_{i1} = c_1 \cdot (p_{i1} - h_1) \cdot (p_{i1} - h_2) \tag{6.4}$$

$$q_{i2} = c_2 \cdot (p_{i2} - v_1) \cdot (p_{i2} - v_2), \tag{6.5}$$

where $c_1, c_2, h_1, h_2, v_1, v_2 \in \mathbb{R}$ are the parameters of the system and i is the index of the oscillator.

The small parameter ϵ endows the system with a slow-fast stucture by separating the timescale of the evolution of p_{i1} and p_{i2} such that p_{i1} is much slower than p_{i2}. For each node, the *fast nullcline* or *critical manifold* is defined as

$$S_o := \{\dot{p}_{i2} = 0\} = \left\{ p_{i2} \left(p_{i1} - c_2 (p_{i2} - v_1)(p_{i2} - v_2) \right) = 0 \right\},$$

and contains two connected components, i.e. the horizontal line $\{p_{i2} = 0\}$ and the parabola $\{p_{i1} - c_2 \cdot (p_{i2} - v_1) \cdot (p_{i2} - v_2) = 0\}$.

By simulating the system, as in Figure 6.1, we can observe that the slow segments of the system trajectories remain in a neighborhood of S_0 past its self-intersection point (dynamic transcritical bifurcation, a bifurcation of the fast system obtained by keeping p_{i1} fixed such that it becomes a parameter in the remaining equation, with p_{i2} as the only variable, discussed for example in Boudjellaba and Sari, 2009), where it becomes repulsive in the normal (p_{i2}) direction. Therefore the solution reacts to this change of attractivity of S_0 with a delay until the fast dynamics takes over, causing "jumps" between the two components of the critical manifold. The strong contraction on the left side of the self-intersection point is what endows this system with robust dynamics, as perturbations to the Voltage-like variable p_{i2} are very quickly cancelled in this region (see Figure 6.2).

The parameters of the simulation in Figure 6.1 are chosen such that the long-term dynamics of the system is oscillatory, with trajectories approaching a limit cycle. This results from the specific arrangement of the fixed points in the system, which give it an excitable structure in the transient dynamics (upon response to the first input spike) and subsequently an oscillatory structure. In fact, if the system is initialized at the only stable

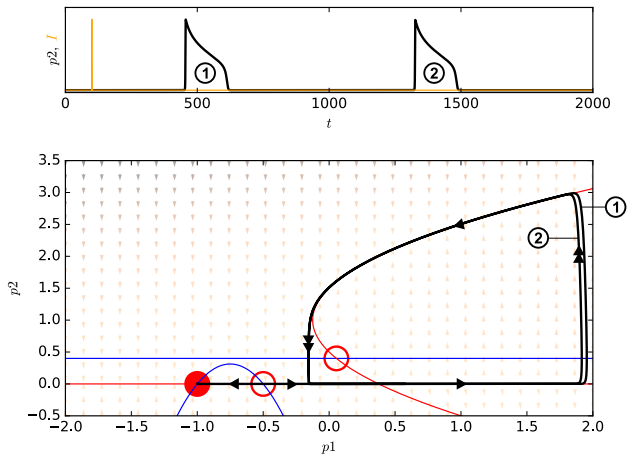


FIGURE 6.1: **Oscillatory regime for planar slow-fast system** described in Equation 6.3. The dynamics of the system is shown for a single node with parameter values ($c_1 = -5$, $h_1 = -1.0$, $h_2 = -0.5$, $c_2 = 0.5$, $v_1 = 0.5$, $v_2 = 1.5$, $\alpha = 0.4$, $\epsilon = 1 \cdot 10^{-3}$) and initial conditions (p_1 = -1.0, p_2 = 0.05). The critical manifold is here shown in red, whereas the slow nullcline is shown in blue. The intersections between these are the fixed points of the system. The two unstable fixed points are shown as empty circles, whereas the stable one is shown as a filled circle. Additionally, we include the vector field (in the background), where darker arrows are associated with points where the vector field has larger magnitude. The system's trajectory is shown as a black curve. A single black triangle on the curve denotes a slow segment in the system's trajectory, whereas a double black triangle denotes a fast one. The state of the system is initialized near a stable fixed point (the filled red dot on the left), and destabilized at time $t = 100$ by an input perturbation I, of amplitude $K = 4$ and duration $s = 0.4$ (in orange). The perturbation force the system's dynamics to escape the basin of attraction of the stable fixed point, and move to the right of the unstable fixed point on the linear component of the critical manifold, at $p_{i2} = 0$. The dynamics slowly move along the linear component of the critical manifold, until they quickly jump on its parabolic component. There, they move slowly along the parabola (but in the inverse p_{i1} direction), finally jumping again towards the linear component of the critical manifold, to the right of the unstable fixed point on this component. The described dynamics is then repeated; the initial perturbation caused the system to enter an oscillatory regime, where the dynamics asymptotically approach a limit cycle. Note that an analytical solution to the limit cycle is not available, although the second oscillation (labelled with a 2 in the Figure) is already very close to it.

equilibrium, and in the absence of any perturbation I, its dynamics just stays there forever. For a small perturbation I, the system is displaced from the stable fixed point, but remains in its basin of attraction, so that the dynamics takes the trajectory back to the stable equilibrium without any oscillatory transition. For a strong enough perturbation I, the trajectory is kicked to the right of the unstable fixed point (which thus works as a threshold), moving towards the oscillatory dynamics and the limit cycle.

It is also possible to choose the system's parameters such that its dynamics is excitable,

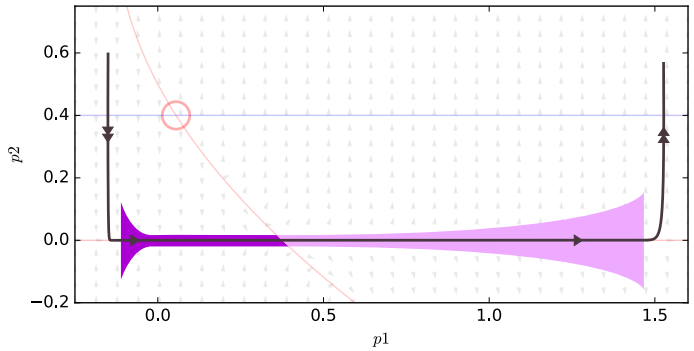


FIGURE 6.2: **Contraction-Expansion of trajectories before and after the dynamic transcritical bifurcation**. The contraction and expansion of trajectories before vs after the dynamic transcritical bifurcation is emphasized by a sand-watch shaped area, where the contraction is shown in purple, and the expansion in pink. An example trajectory is also shown, where we use a double triangle symbol to denote its fast segment, and a single triangle symbol to denote its slow segment.

but not oscillatory. In Figure 6.3, we show the results of a simulation for such a parameter set. In particular, the fixed points in the system can be arranged such that excitability is preserved, (i.e. a strong enough perturbation for a system on the stable fixed point moves its state to the right of an unstable fixed point, causing an oscillation, as shown in Figure 6.3), but the dynamics is not periodic anymore. In fact, as shown in the Figure, after an oscillation, the dynamics returns to the basin of attraction of the system's fixed point, approaching it and thus settling towards a "resting state".

Both of these regimes could give rise to clustered periodicities in a network of coupled slow-fast oscillators, and thus deserve attention in future work.

6.2 Preliminary results

Importantly, preliminary simulations with a system of $N = 5$ slow-fast oscillators as defined in Equation 6.3 show promising results for the re-implementation of the system discussed in Chapter 5. In fact, for initial conditions with de-synchronized oscillators, we observed an attracting synchronized state akin to that of the $N = 5$ system of delay pulse-coupled LIF oscillators (see Figure 6.4).

Furthermore, we were also able to observe (for a slightly different parameter set) perturbation-driven de- and re-synchronization (see Figure 6.5). Because of the timescale separation

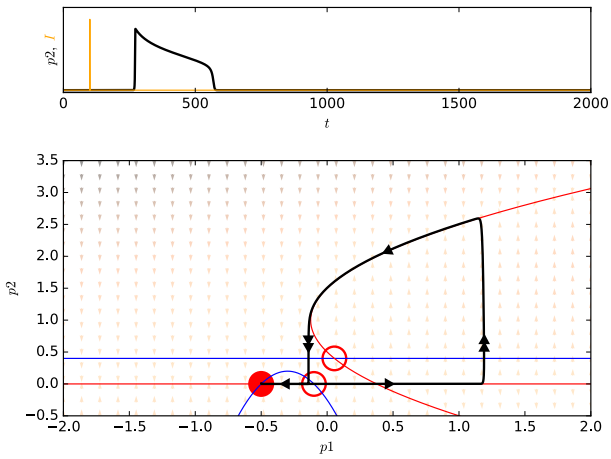


FIGURE 6.3: **Purely excitable regime for planar slow-fast system** in Equation 6.3. The dynamics of the system is shown for a single node with parameter values ($c_1 = -5,\ h_1 = -0.5,\ h_2 = -0.1,\ c_2 = 0.5,\ v_1 = 0.5,\ v_2 = 1.5,\ \alpha = 0.4,\ \epsilon = 1 \cdot 10^{-3}$) and initial conditions (p_1 = -0.5, p_2 = 0.05). The state of the system is initialized near a stable fixed point (the filled red circle on the left), and destabilized by an input perturbation I at time $t = 100$ (of amplitude $K = 4$ and duration $s = 0.4$). The perturbation causes the state of the system to move away from the basin of attraction of the stable fixed point, to the right of the unstable fixed point on the $p_{i2} = 0$ component of the critical manifold (the empty red circle on the line $p_{i2} = 0$). The dynamics slowly move along the linear component of the critical manifold, until they quickly jump on its parabolic component. There, they move slowly along the parabola (in the inverse p_{i1} direction), finally jumping again towards the linear component of the critical manifold and returning in the basin of attraction of the stable fixed point, which the dynamics then asymptotically approach; no further oscillations are thus possible in the absence of input perturbations.

and its dimension, the system is quite stiff;[1] for this reason, we decided to collect additional numerical evidence for the existence of the synchronized states found in the simulations in Figure 6.5. To do so we performed a periodic continuation in ϵ for the clustered states (a, b, b, c, c), (a, b, a, c, c) and (a, a, b, c, c), which we report in Figure 6.6 for two solution measures. Given that the solver converges at each step of the continuation, we believe that these results give good evidence that the solutions are truly periodic, and that they coexist in the parameter set considered in our simulations. Assessing the stability of these synchronized states is not trivial, given the stiffness of the system; nevertheless, the state transitions observed in the simulations shown in Figure 6.5 provide reasonable evidence for the existence of attractive directions.

Moreover, the simulations bring numerical evidence that transitions between these states,

[1] We simulated the system with different libraries and solvers, and encountered numerical instability with some of the solvers used.

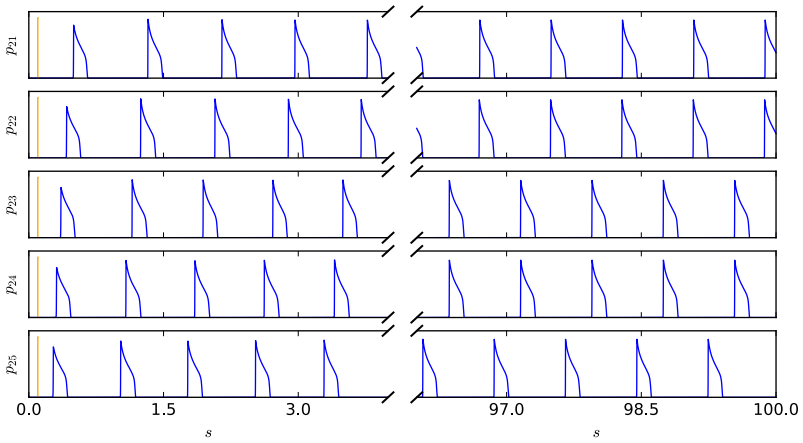


FIGURE 6.4: **Clustered synchronization in slow-fast system**. The dynamics of a system of 5 slow-fast oscillators as described in Equation 6.3 is shown (time is rescaled as $s = \epsilon t$, also known as *fast time*). Here the system parameters are set as ($c_1 = -5,\ h_1 = -1.0,\ h_2 = -0.5,\ c_2 = 0.5,\ v_1 = 0.5,\ v_2 = 1.5,\ \alpha = 0.4,\ \epsilon = 1 \cdot 10^{-3},\ g = 1 \cdot 10^{-4}$), and initial conditions as ($p_{11} = -1.2, p_{21} = -1.1, p_{31} - 1.0, p_{41} = -0.9, p_{51} = -0.8$) and $p_{i2} = 0.05$ for $i = 1, \ldots, 5$. An input perturbation I of amplitude $K = 4$ and duration $d = 4 \cdot 10^{-4}$ is applied at time $s = 0.1$ to all oscillators.

and even cycles are possible in this new slow-fast framework, akin to those already characterized in the heteroclinic paradigm. We thus believe we will be able in future work to further characterize periodicities (and associated clustered states) in parameter space, by means of e.g. numerical continuation, and search for other synchronized states in the phase space, thus verifying if it is indeed possible to re-implement heteroclinic network computation (such as the one described in Chapter 5) with a fundamentally different kind of dynamics, which presents several advantages over the heteroclinic dynamics framework, as previously discussed.

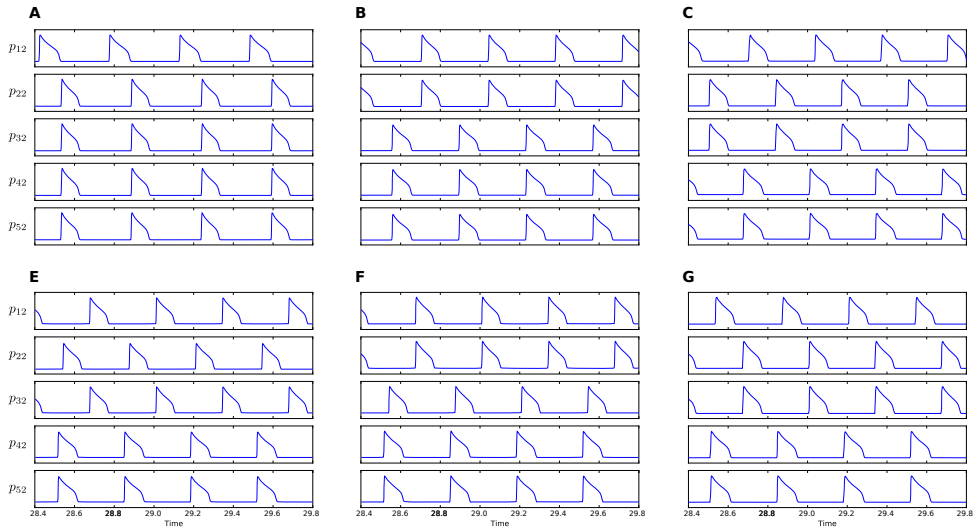


FIGURE 6.5: **Perturbation-driven synchronized state transitions in slow-fast system**, (time rescaled as $s = \epsilon t$). We simulate the system for parameter set ($c_1 = -5$, $h_1 = -1.3$, $h_2 = -1.1$, $c_2 = 0.5$, $v_1 = 0.5$, $v_2 = 1.5$, $\alpha = 0.4$) and initial conditions $p_{i1} = -1.3, p_{i2} = 0.005$ for all i. We first apply an input pulse to p_{12} of amplitude $K = 15$ and duration $d = 1.5 \cdot 10^{-3}$ to p_{12} at time $t = 1$, causing a de-synchronization of the first oscillator, leading to a synchronized state (a, b, b, b, b) (panel A). We start a new simulation where the initial conditions are the last state p_{i1}, p_{i2} with $i = 1, \ldots, 5$ of the previous simulation, and perturb oscillator 2 with another pulse (same parameters as before), leading to a synchronized state (a, a, b, b, b) (panel B). We now perturb oscillator 3, leading to a synchronized state (a, b, b, c, c) (panel C). A perturbation to oscillator 1 leads to a state (a, b, a, c, c) (panel D), then to a state (a, a, b, c, c) when oscillator 2 is perturbed (panel E) and finally to a state (a, b, b, c, c) when oscillator 3 is perturbed (panel F). These simulations shows that not only the slow-fast system presents synchronized states, but these are also connected, and they can form cycles (the synchronized states in panel C ,D, E, F).

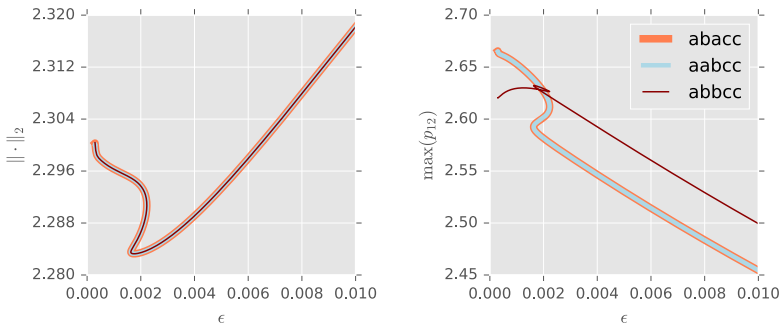


FIGURE 6.6: **Periodic continuation in ϵ of synchronized states found in simulation.** Data from one period of the orbit in each of the synchronized states (a, b, a, c, c), (a, a, b, c, c) and (a, b, b, c, c) found in the simulations reported in Figure 6.5 is used as an initial guess for the periodic continuation. We show two solution measures for the continuation, an L2 norm measure on the left, and $\max(p_{12})$ on the right. The convergence of the solver at each step gives evidence that these states are in fact synchronized.

Chapter 7

Conclusions and future directions

In the Introduction to this thesis we purported to explore the interface between symbolic models of computation and dynamical system models of neural dynamics. In characterizing computation in such models, we adopted a restricted form of the framework proposed by MacLennan (2004), where we say that a dynamical system performs a computation if a correspondence can be found between it and a formal system.[1] The broader context of the contributions presented in this thesis, i.e. the long term goal inspiring this exploration, is thus to attain a correspondence between suitable formal systems and the dynamics of real neural circuits.

In order to do so, we have first introduced the Versatile Shift, a formal system which allows for the parsimonious real-time simulation of a range of models of computation, and which Gödelization defines Nonlinear Dynamical Automata, a class of piecewise affine-linear dynamical systems. Simulable models include, for example, Finite-State Machines, Push-Down Automata, Top-Down Recognizers and Turing Machines. We have then defined a constructive mapping between Nonlinear Dynamical Automata and Recurrent Neural Networks, thus allowing the realization of Versatile Shifts (and the models they can simulate)

[1] The framework in MacLennan (2004) is in fact more broadly defined, where a physical system is said to perform a computation if a perfect or approximate correspondence can be found between it and an abstract mathematical object (not necessarily a formal system), and this correspondence has a use in the wider context which the system is a part of (that is, realizing said abstract object can be seen as the system's purpose). In this thesis, we ignored issues of purpose for the systems studied, as we characterized their dynamics in isolation and in the abstract.

in the form of RNN dynamics. The Neural Networks we construct implement the original symbolic computation parsimoniously, transparently and modularly. Crucially, the granular modularity of the network architecture defined by the mapping has important consequences for the implementation of interactive models of computation, distinguishing our contribution from previous work. Furthermore, we presented initial evidence that the constructive mapping could lead in future work to the possibility of correlational studies with, e.g., Event-Related Potential measures from large-scale brain recordings.

Mapping formal systems to RNN dynamics in order to compare observables with those from real neural systems suffers from severe limitations. In fact, the level of description of RNNs is quite removed from that required to characterize computation in real neural systems. In Chapter 5 we thus moved our attention to the characterization of computation in a class of continuous-time neural models, moving a step closer to the desired level of abstraction. Specifically, we showed that noiseless systems of neural oscillators supporting the emergence of heteroclinic networks (as characterized by Neves and Timme, 2009, 2012) can be seen as realizations of Finite-State Transducers. We have also shown that, when noise is present, the state switching they implement becomes probabilistic, with the probability distribution of the possible transitions at each state smoothly flattening as a noise strength parameter is increased. That is, for noise strength approaching zero, the probability of one of the transitions approaches 1; for increasing noise, other transitions become more and more likely, approaching a uniform distribution over the transitions. Furthermore, we have shown that for intermediate levels of noise, the amount of information that the system is able to transmit increases. This is a key result in relation to the modelling of biological systems that are hypothesized to implement heteroclinic computation; we now know that such systems could harvest noise to promote information processing, consistently with results from the field of stochastic facilitation (Magalhães and Kohn, 2011, McDonnell and Ward, 2011, Wiesenfeld et al., 1995, Zeng et al., 2000). In order to formally characterize computation in noisy heteroclinic networks, future work will see the definition of a correspondence between these systems and probabilistic Finite-State Transducers (or similar models based on a Markov Chain abstraction).

Systems supporting the emergence of heteroclinic networks also suffer from important

drawbacks that limit the possibility of relating their dynamics to the dynamics of real neural systems. In particular, persistent heteroclinic switching in these systems is only possible when input and noise perturbations have extremely small magnitude (as in the system discussed in Chapter 5). While efforts are being made to increase stability in the heteroclinic computing framework (e.g. the possibility of Stable Heteroclinic Channels, as in Rabinovich et al., 2008b), this is proving to be an elusive objective. Additionally, in the noiseless heteroclinic system, the transition between states is always asymptotic. That is, an infinite amount of time is needed for the system to complete a transition between one saddle state and the next.

To attain the possibility of detailed correlational studies with observables from real neural circuits, we must thus look even further, by turning to the class of models that most closely relate to the level of description of electrophysiological measurements, i.e. smooth excitable neural circuit models (such as the well-known neural model proposed by Hodgkin and Huxley, 1952). As discussed in Chapter 6, slow-fast dynamical systems are interesting candidates in this direction, as preliminary numerical results suggest they can support non-asymptotic and robust state-switching dynamics, overcoming the drawbacks of heteroclinic systems.

The definition of a constructive mapping between a formal system and a smooth excitable neural circuit model would allow us to effectively explore the extent to which the dynamics of real neuronal systems could be modeled in terms of symbolic computation (see Figure 7.1). This exploration can be guided by the use of Machine Learning techniques, as we will discuss in Section 7.2.

To summarize, in this work we presented the following key contributions:

– Introduction of the *Versatile Shift*, a formal system which can efficiently simulate a range of models of computation, and which presents important similarities with the action of rewriting systems, leading to promising possibilities in future work.

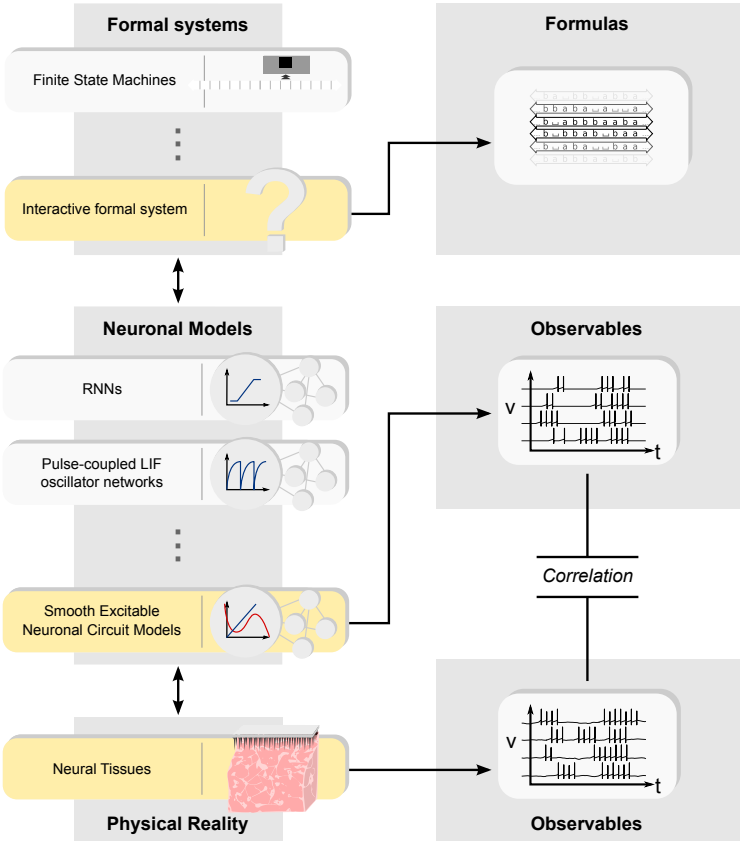


FIGURE 7.1: **Characterization of computation in neural tissues through smooth excitable neural models**. In order to characterize computation in neural tissues, we believe a constructive mapping must be devised between a formal system and smooth excitable neural circuit models, i.e. the class of neural models that most closely relate to the level of description of electrophysiological measures of neural activity. In particular, the formal system should capture notions of interactive computation (as discussed in Section 7.1), as a neural tissue is in constant interaction with other neural assemblies and possibly the environment, and this interaction is a defining characteristic of its dynamics.

– Definition of a constructive mapping between Versatile Shifts and Nonlinear Dynamical Automata through Gödelization, allowing for the mapping of string rewriting on a symbolic space to piecewise affine-linear dynamics on a vectorial space.

– Definition of a transparent and modular network architecture simulating Nonlinear Dynamical Automaton dynamics on the unit square through the evolution of neural activations in time. As we have shown that Nonlinear Dynamical Automata can simulate Versatile Shifts efficiently, the presented architecture is thus able to simulate Versatile Shifts efficiently.

– Identification of a facilitatory effect of noise on the Finite State computation performed by class of neural systems which can support the dynamic switching of states (State-Switching Machines). If a general constructive mapping between formal systems and biologically plausible neuronal systems can be found, it is crucial to understand how noise (undefined in formal systems) can affect the symbolic computation supported by suitable neural systems. These results suggests that noise can actually facilitate computation in these systems, under the right conditions.

– Presentation of preliminary results suggesting the possibility of robust dynamical switching of states in slow-fast systems, which, if confirmed in future work, would mean the possibility of robust Finite State Machine computation in a class of models of special biological relevance.

While the advancements outlined in this thesis bring us closer to our goal, the definition of a general constructive mapping between formal systems and smooth excitable circuit models is not yet within our reach, although we hope that this work will contribute to the laying of the necessary foundations that will make the endeavour possible.

7.1 Formalizing interaction in dynamical systems

Computation in real neuronal circuits is carried out in the context of a continuous interaction with the environment and other neural circuits. It is therefore crucial, when characterizing their computation in terms of some formal system, that the chosen system can capture notions of interactivity (see Milner et al., 1992 for an example of a purely formal system that can model interactions).

In Chapter 4 we have presented two examples of interactive neural systems, as we have shown how to construct a RNN implementation of i) a Central Pattern Generator modelled as a Finite-State Machine, and ii) an Interactive Automata Network. In the first case, we (implicitly) used the Finite-State Machine as an interactive model of computation, by considering its output to be the sequence of states induced by a continuous input stream.

In the second, we allowed communication between automata without defining a matching mechanism in Versatile Shifts (VSs).

In both instances, we deliberately ignored the fact that our methods do not define a way in which Versatile Shifts and, consequently, NDA, can interact with the external world. Therefore, we aim in future work to formally characterize interaction at two levels in our methodology, that is, at the symbolic the level of the VS, and at the vectorial level of NDA. Additionally, we must define the relation between interactions at the two level, in order to maintain the possibility of mapping one to the other. Critically, beim Graben (2008) has shown that interactive computation in NDA can be modelled through the definition of "quantum operators", where the interactive machine's configuration is represented in the NDA as a point on the unit square, whereas incoming input is mapped to functions modifying the landscape of the vector space on which the NDA evolves, thus transforming the NDA state compatibly with the symbologram representation of the input-perturbed machine configuration (see Figure 7.2). Future work will see the integration of this approach with the methods presented in Chapter 4.

It is important to point out that, in order to attain a constructive mapping between formal systems and dynamical models of excitable neural circuits (as envisioned in Figure 7.1), or any class of continuous-time dynamical systems for that matter, the seminal work by beim Graben (2008) formalizing interaction in NDA must be extended to account for continuous time interaction with the environment. This could eventually inspire the creation of a general framework for the characterization of computation in open physical systems. Importantly, this could also lead to a greater understanding of the interactive computation performed by Machine Learning models such as Liquid State Machines (Maass et al., 2002), which have proved to be especially powerful models, yet especially obscure with regards to how the computation they perform is implemented.

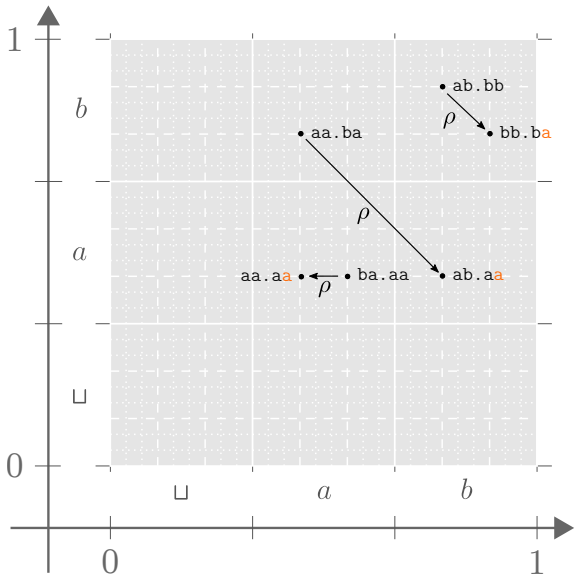


FIGURE 7.2: **Quantum operator acting on the symbologram representation of a bounded queue.** The symbologram encodes the contents of a first-in-first-out (FIFO) queue of capacity 4, where incoming input symbols added at the front of the queue force the queue to "drop" symbols at its back when the queue is full. On the symbologram, the reception of an input symbol `a` is equivalent to the application of a "quantum operator" ρ (beim Graben, 2008) to the encoded queue, which maps each encoded queue configuration to a new encoded input-perturbed configuration. In the Figure we show how the ρ operator transforms 3 points on the symbologram, where each point is associated to its corresponding symbolic sequence, and where we highlight the incoming input symbol `a` in orange.

7.2 Machine Learning of physical computation

A constructive mapping between a suitable interactive formal system and excitable dynamical system models of neural circuits would allow us to characterize the computation performed by real neuronal assemblies. In fact, given electrophysiological measurements from the activity of a neural circuit, and a hypothesis (or hypothesis set) towards the specific computations that the circuit might be performing, we would be able to construct an excitable model realizing the computation and then explore possible correlations between its observables and the measurements from the real system (see Figure 7.1). However, the hypothesis set to explore can be huge, as the same computation can be performed in an infinite number of ways by the formal system, leading to an infinite number of possible dynamical systems that realize the computation.

The search process could be automated by using Machine Learning techniques such as

symbolic regression and genetic programming (Koza, 1994, Schmidt and Lipson, 2009). The learning algorithm would start from random guesses in the hypothesis space for the computation performed in the real physical system, by randomly composing primitives in the formal system. Given that a constructive mapping to excitable systems is provided, the algorithm would derive the dynamical system from its guesses (i.e. specific instances of the formal system), and compare its dynamics with the real dynamics of the physical system. By an evolutionary selection mechanism, the best guesses (i.e. the ones that lead to better "fit" between the generated dynamics and the dynamics to be modelled) would be used at the next step to generate new guesses, which are then compared based on the goodness of fit of the dynamical systems they define, and so forth. The search space could be further restricted through the addition of appropriate constraints (e.g. notions of parsimony for the generated formal systems).

Such an algorithm would automatically search the space of possible formal systems that can be used to describe the computation performed by the real neural circuit, and return a set of "best guesses". Of course, once suitable formal systems are found, it is up to the scientist to specify reasonable "semantics" for the formal systems guessed by the algorithm, which could be a tricky endeavour itself, depending on the complexity of the system's description.

Note that the procedure we outline here is not restricted to excitable neural circuits. Given any class of physical systems that can be seen as performing a computation, and a constructive mapping between a formal system and a model of the physical system dynamics, then symbolic regression could be applied to automatically characterize the computation performed by the physical system in terms of the defined formal system. Importantly, if the formal system in the constructive mapping is interactive, and operators can be defined that map interactions in the formal systems to their representation in the physical system model (as discussed in Section 7.1), then it will also be possible to characterize the computation performed by open physical systems. This is especially relevant for the field of Artificial Intelligence, which is interested in characterizing computation in intelligent systems, an important class of open physical systems. Crucially, research in Machine Learning has for a long time adopted the classical approach to computation as an input-output process; recent developments, however, reflecting similar considerations emerged in contemporary

trends in the Theory of Computation (Goldin et al., 2006, Wegner, 1998), have highlighted the limits of such approach (Cristianini, 2010). It is now clear that any general theory of intelligent behaviour, if one can be found, must pursue the formalization of the concept of interaction in intelligent systems, and its role in the computations they realize.

Bibliography

Afraimovich, V. S., Rabinovich, M. I., and Varona, P. (2004). Heteroclinic contours in neural ensembles and the winnerless competition principle. *International Journal of Bifurcation and Chaos*, 14(04):1195–1208.

Aho, A. V. and Ullman, J. D. (1972). *The theory of parsing, translation, and compiling.* Prentice-Hall, Inc.

Alvarez-Alvarez, A., Trivino, G., and Cordón, O. (2012). Human gait modeling using a genetic fuzzy finite state machine. *Fuzzy Systems, IEEE Transactions on*, 20(2):205–223.

Amari, S.-I. (1974). A method of statistical neurodynamics. *Kybernetik*, 14:201 – 215.

Ashwin, P. and Borresen, J. (2005). Discrete computation using a perturbed heteroclinic network. *Phys. Lett. A*, 374(4–6):208–214.

Ashwin, P. and Marc, T. (2005a). Unstable attractors: existence and robustness in networks of oscillators with delayed pulse coupling. *Nonlinearity*, 18:2035–2060.

Ashwin, P. and Marc, T. (2005b). When instability makes sense. *Nature*, 436:36–37.

Ashwin, P. and Postlethwaite, C. (2013). On designing heteroclinic networks from graphs. *Physica D: Nonlinear Phenomena*, 265:26–39.

Barrès, V., III, A. S., and Arbib, M. (2013). Synthetic event-related potentials: A computational bridge between neurolinguistic models and experiments. *Neural Networks*, 37:66 – 92.

beim Graben, P. (2008). Quantum Representation Theory for Nonlinear Dynamical Automata. In *Advances in Cognitive Neurodynamics ICCN 2007*, pages 469–473. Springer.

beim Graben, P. and Drenhaus, H. (2012). Computationelle Neurolinguistik. *Zeitschrift für Germanistische Linguistik*, 40(1):97 – 125.

beim Graben, P., Gerth, S., and Vasishth, S. (2008). Towards dynamical system models of language-related brain potentials. *Cognitive Neurodynamics*, 2(3):229–255.

beim Graben, P., Jurish, B., Saddy, D., and Frisch, S. (2004). Language processing by dynamical systems. *International Journal of Bifurcation and Chaos*, 14(02):599–621.

beim Graben, P. and Potthast, R. (2014). Universal neural field computation. In *Neural Fields*, pages 299–318. Springer.

beim Graben, P. and Rodrigues, S. (2013). A biophysical observation model for field potentials of networks of leaky integrate-and-fire neurons. *Frontiers in Computational Neuroscience*, 6(100).

Benoît, E. (1991). *Dynamic bifurcations: proceedings of a conference held in Luminy, France, March 5-10, 1990*. Number 1493. Springer Verlag.

Bick, C. and Rabinovich, M. (2009). Dynamical origin of the effective storage capacity in the brain's working memory. *Phys. Rev. Lett.*, 103:218101.

Borisyuk, R. M. and Borisyuk, G. N. (1997). Information coding on the basis of synchronization of neuronal activity. *BioSystems*, 40(1):3–10.

Boudjellaba, H. and Sari, T. (2009). Dynamic transcritical bifurcations in a class of slow–fast predator–prey models. *Journal of Differential Equations*, 246(6):2205–2225.

Bournez, O. and Campagnolo, M. L. (2008). A survey on continuous time computations. In *New Computational Paradigms*, pages 383–423. Springer.

Branicky, M. S. (1995). Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, 138(1):67–100.

Burgin, M. (2003). Information theory: A multifaceted model of information. *Entropy*, 5(2):146–160.

Burgin, M. (2010). *Theory of information: fundamentality, diversity and unification*, volume 1. World Scientific.

Cabessa, J. and Siegelmann, H. T. (2012). The computational power of interactive recurrent neural networks. *Neural Computation*, 24(4):996–1019.

Cabessa, J. and Villa, A. E. (2012). The expressive power of analog recurrent neural networks on infinite input streams. *Theoretical Computer Science*, 436:23–34.

Cabessa, J. and Villa, A. E. (2013). The super-turing computational power of interactive evolving recurrent neural networks. In *Artificial Neural Networks and Machine Learning–ICANN 2013*, pages 58–65. Springer.

Carmantini, G. (2016). Youtube video. `https://youtu.be/cPaYNGmUYTk`.

Carmantini, G. S., beim Graben, P., Desroches, M., and Rodrigues, S. (2015). Turing computation with recurrent artificial neural networks. In *Proceedings of the NIPS Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches*, pages 5 – 13. arXiv:1511.01427 [cs.NE].

Carmantini, G. S., beim Graben, P., Desroches, M., and Rodrigues, S. (2016). A modular architecture for transparent computation in recurrent neural networks. *Neural Networks*.

Chakrapani, L. N., Korkmaz, P., Akgul, B. E., and Palem, K. V. (2007). Probabilistic system-on-a-chip architectures. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 12(3):29.

Chklovskii, D. B., Mel, B., and Svoboda, K. (2004). Cortical rewiring and information storage. *Nature*, 431(7010):782–788.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124.

Collins, J. J. and Richmond, S. A. (1994). Hard-wired central pattern generators for quadrupedal locomotion. *Biological Cybernetics*, 71(5):375 – 385.

Collins, S. H. and Ruina, A. (2005). A bipedal walking robot with efficient and human-like gait. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 1983–1988. IEEE.

Copeland, B. J. (2002). Hypercomputation. *Minds and machines*, 12(4):461–502.

Cristianini, N. (2010). Are we there yet? *Neural Networks*, 23(4):466–470.

Dambre, J., Verstraeten, D., Schrauwen, B., and Massar, S. (2012). Information processing capacity of dynamical systems. *Scientific reports*, 2.

Desroches, M., Guckenheimer, J., Krauskopf, B., Kuehn, C., Osinga, H. M., and Wechselberger, M. (2012). Mixed-mode oscillations with multiple time scales. *SIAM Review*, 54(2):211–288.

Dodig-Crnkovic, G. (2011). Significance of models of computation, from turing model to natural computation. *Minds and Machines*, 21(2):301–322.

Emerson, R. C., Bergen, J. R., and Adelson, E. H. (1992). Directionally selective complex cells and the computation of motion energy in cat visual cortex. *Vision research*, 32(2):203–218.

Frank, S. L., Otten, L. J., Galli, G., and Vigliocco, G. (2015). The ERP response to the amount of information conveyed by words in sentences. *Brain and Language*, 140:1 – 11.

Frisch, S., beim Graben, P., and Schlesewsky, M. (2004). Parallelizing grammatical functions: P600 and P345 reflect different cost of reanalysis. *International Journal of Bifurcation and Chaos*, 14(2):531 – 549.

Glendinning, P. (1994). *Stability, instability and chaos: an introduction to the theory of nonlinear differential equations*, volume 11. Cambridge university press.

Gödel, K. (1931). Über formal unentscheidbare sätze der *principia mathematica* und verwandter systeme I. *Monatshefte für Mathematik und Physik*, 38:173 – 198.

Goldin, D., Smolka, S. A., and Wegner, P. (2006). *Interactive computation*. Springer.

Golubitsky, M., Stewart, I., Buono, P.-L., and Collins, J. J. (1998). A modular network for legged locomotion. *Physica D*, 115(1-2):56 – 72.

Golubitsky, M., Stewart, I., Buono, P.-L., and Collins, J. J. (1999). Symmetry in locomotor central pattern generators and animal gaits. *Nature*, 401(6754):693 – 695.

Hodgkin, A. and Huxley, A. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol.*, (117):500–544.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison–Wesley, Menlo Park, California.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558.

Horchler, A. D., Daltorio, K. A., Chiel, H. J., and Quinn, R. D. (2015). Designing responsive pattern generators: stable heteroclinic channel cycles for modeling and control. *Bioinspiration & biomimetics*, 10(2):026001.

Huerta, R., Nowotny, T., García-Sanchez, M., Abarbanel, H. D., and Rabinovich, M. I. (2004). Learning classification in the olfactory system of insects. *Neural computation*, 16(8):1601–1640.

Ijspeert, A. J. (2008). Central pattern generators for locomotion control in animals and robots: a review. *Neural Networks*, 21(4):642–653.

Jaeger, H. and Haas, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80.

Karttunen, L. (2000). Applications of finite-state transducers in natural language processing. In *International Conference on Implementation and Application of Automata*, pages 34–46. Springer.

Kirst, C., Geisel, T., and Timme, M. (2009). Sequential desynchronization in networks of spikin neurons with partial resets. *Phys. Rev. Lett.*, 102:068101.

Kleene, S. (1956). Neural nets and automata. *Automata Studies*, pages 3–43.

Koza, J. R. (1994). Genetic programming ii: Automatic discovery of reusable subprograms. *Cambridge, MA, USA*.

Krupa, M. (1997). Robust heteroclinic cycles. *J. Nonlinear Sci.*, 7:129–176.

Langton, C. G. (1990). Computation at the edge of chaos: phase transitions and emergent computation. *Physica D: Nonlinear Phenomena*, 42(1):12–37.

Larger, L., Soriano, M. C., Brunner, D., Appeltant, L., Gutierrez, J. M., Pesquera, L., Mirasso, C. R., and Fischer, I. (2012). Photonic information processing beyond turing: an optoelectronic implementation of reservoir computing. *Optics Express*, 20(3):3241–3249.

Laurent, G., Stopfer, M., Friedrich, R. W., Rabinovich, M. I., Volkovskii, A., and Abarbanel, H. D. (2001). Odor encoding as an active, dynamical process: experiments, computation, and theory. *Annual review of neuroscience*, 24(1):263–297.

Lewis, R. L. (1998). Reanalysis and limited repair parsing: Leaping off the garden path. In Fodor, J. D. and Ferreira, F., editors, *Reanalysis in Sentence Processing*, pages 247 – 285. Kluwer.

Lind, D. and Marcus, B. (1995). *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, Cambridge (UK). Reprint 1999.

LukošEvičIus, M. and Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149.

Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560.

MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

MacLennan, B. J. (2004). Natural computation and non-turing models of computation. *Theoretical computer science*, 317(1):115–145.

Magalhães, F. H. and Kohn, A. F. (2011). Vibratory noise to the fingertip enhances balance improvement associated with light touch. *Experimental brain research*, 209(1):139–151.

Mazor, O. and Laurent, G. (2005). Transient dynamics versus fixed points in odor representations by locust antennal lobe projection neurons. *Neuron*, 48:661–673.

McCulloch, W. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133.

McDonnell, M. D. and Ward, L. M. (2011). The benefits of noise in neural systems: bridging theory and experiment. *Nature Reviews Neuroscience*, 12(7):415–426.

McGhee, R. B. (1968). Some finite state aspects of legged locomotion. *Mathematical Biosciences*, 2(1-2):67–84.

Milner, R., Parrow, J., and Walker, D. (1992). A calculus of mobile processes, pt.1. *Information and computation*, 100(1):1–40.

Minsky, M. L. (1967). *Computation: finite and infinite machines*. Prentice-Hall, Inc.

Mirollo, R. E. and Strogatz, S. H. (1990). Synchronization of pulse-coupled biological oscillators. *SIAM Journal on Applied Mathematics*, 50(6):1645–1662.

Moore, C. (1990). Unpredictability and undecidability in dynamical systems. *Physical Review Letters*, 64(20):2354.

Moore, C. (1991). Generalized shifts: unpredictability and undecidability in dynamical systems. *Nonlinearity*, 4(2):199.

Näätänen, R. (1990). The role of attention in auditory information processing as revealed by event-related potentials and other brain measures of cognitive function. *Behavioral and Brain Sciences*, 13(02):201–233.

Neves, F. S. (2010). *Universal Computation and Memory by Neural Switching*. PhD thesis, Division of Mathematics and Natural Sciences Georg-August-University, Göttingen.

Neves, F. S. and Timme, M. (2009). Controlled perturbation-induced switching in pulse-coupled oscillator networks. *J. Phys. A*, 42(34):345103.

Neves, F. S. and Timme, M. (2012). Computation by switching in complex networks of states. *Phys. Rev. Lett.*, 109(1):018701.

Orponen, P. (1997). A survey of continuous-time computation theory. In *Advances in algorithms, languages, and complexity*, pages 209–224. Springer.

Osterhout, L., Holcomb, P. J., and Swinney, D. A. (1994). Brain potentials elicited by garden-path sentences: Evidence of the application of verb information during parsing. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 20(4):786 – 803.

Palem, K. V. (2005). Energy aware computing through probabilistic switching: A study of limits. *Computers, IEEE Transactions on*, 54(9):1123–1137.

Rabinovich, M., Huerta, R., and Laurent, G. (2008a). Transient dynamics for neural processing. *Science*, 321:48–50.

Rabinovich, M. I., Huerta, R., Varona, P., and Afraimovich, V. S. (2008b). Transient cognitive dynamics, metastability, and decision making. *Plos Computational Biology*, 4(5):e1000072.

Rodrigues, S., Desroches, M., Krupa, M., Cortes, J. M., Sejnowski, T. J., and Ali, A. B. (2016). Time-coded neurotransmitter release at excitatory and inhibitory synapses. *Proceedings of the National Academy of Sciences*, 113(8):E1108–E1115.

Schmidt, M. and Lipson, H. (2009). Distilling free-form natural laws from experimental data. *science*, 324(5923):81–85.

Schöner, G., Jiang, W. Y., and Kelso, J. A. S. (1990). A synergetic theory of quadrupedal gaits and gait transitions. *Journal of Theoretical Biology*, 142(3):359 – 391.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423.

Shik, M. L., Severin, F. V., and Orlovsky, G. N. (1966). Control of walking and running by means of electrical stimulation of mid-brain. *BIOPHYSICS-USSR*, 11(4):756.

Siegelmann, H. T. (1995). Computation beyond the turing limit. *Science*, 268(5210):545–548.

Siegelmann, H. T. and Fishman, S. (1998). Analog computation with dynamical systems. *Physica D: Nonlinear Phenomena*, 120(1):214–235.

Siegelmann, H. T. and Sontag, E. D. (1991). Turing computability with neural nets. *Appl. Math. Lett*, 4(6):77–80.

Siegelmann, H. T. and Sontag, E. D. (1995). On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150.

Sipser, M. (2006). *Introduction to the Theory of Computation*. Thomson Course Technology Boston.

Smith, J. C., Abdala, A., Koizumi, H., Rybak, I. A., and Paton, J. F. (2007). Spatial and functional architecture of the mammalian brain stem respiratory network: a hierarchy of three oscillatory mechanisms. *Journal of neurophysiology*, 98(6):3370–3387.

Smith, J. C., Abdala, A. P., Borgmann, A., Rybak, I. A., and Paton, J. F. (2013). Brainstem respiratory networks: building blocks and microcircuits. *Trends in neurosciences*, 36(3):152–162.

Smith, J. E. and Nair, R. (2005). The architecture of virtual machines. *Computer*, 38(5):32–38.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In Rumelhart, D. E., McClelland, J. L., and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume I, chapter 6, pages 194 – 281. MIT Press, Cambridge (MA).

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46(1-2):159 – 216.

Smullyan, R. M. (1961). *Theory of formal systems*. Princeton University Press.

Spröwitz, A., Moeckel, R., Vespignani, M., Bonardi, S., and Ijspeert, A. J. (2014). Roombots: A hardware perspective on 3d self-reconfiguration and locomotion with a homogeneous modular robot. *Robotics and Autonomous Systems*, 62(7):1016–1033.

Stepney, S. (2012). Nonclassical computation – dynamical systems perspective. In *Handbook of natural computing*, pages 1979–2025. Springer.

Strogatz, S. H. (2014). *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. Westview press.

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

Tabor, W. (2009). A dynamical systems perspective on the relationship between symbolic and non-symbolic computation. *Cognitive Neurodynamics*, 3(4):415 – 427.

Tabor, W., Cho, P. W., and Szkudlarek, E. (2013). Fractal analyis illuminates the form of connectionist structural gradualness. *Topics in Cognitive Science*, 5:634 – 667.

Tabor, W., Juliano, C., and Tanenhaus, M. K. (1997). Parsing in a dynamical system: An attractor-based account of the interaction of lexical and structural constraints in sentence processing. *Language and Cognitive Processes*, 12(2/3):211 – 271.

Timme, M., Wolf, F., and Geisel, T. (2003). Unstable attractors induce perpetual synchronization and desynchronization. *Chaos*, 13(1):377–387.

Turing, A. M. (1937). On computable numbers, with an application to the *Entscheidungsproblem. Proc. London Math. Soc*, 42.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, pages 433–460.

Wegner, P. (1998). Interactive foundations of computing. *Theoretical Computer Science*, 192:315 – 351.

Wiesenfeld, K., Moss, F., et al. (1995). Stochastic resonance and the benefits of noise: from ice ages to crayfish and squids. *Nature*, 373(6509):33–36.

Wiggins, S. (2003). *Introduction to applied nonlinear dynamical systems and chaos*, volume 2. Springer Science & Business Media.

Wordsworth, J. and Ashwin, P. (2008). Spatiotemporal coding of inputs for a system of globally coupled phase oscillators. *Phys. Rev. E*, 78:066203.

Zeng, F.-G., Fu, Q.-J., and Morse, R. (2000). Human hearing enhanced by noise. *Brain research*, 869(1):251–255.

Appendix

This appendix contains two papers published as a result of the research underlying this thesis, respectively titled "Turing Computation with Recurrent Artificial Neural Networks" (Carmantini et al., 2015), and "A modular architecture for transparent computation in Recurrent Neural Networks"(Carmantini et al., 2016).

Turing Computation with Recurrent Artificial Neural Networks

Giovanni S. Carmantini*[1], Peter beim Graben[2], Mathieu Desroches[3] and Serafim Rodrigues[1]

[1]School of Computing, Electronics and Mathematics, Plymouth University, United Kingdom
[2]Bernstein Center for Computational Neuroscience Berlin, Humboldt-Universität zu Berlin, Germany
[3]Inria Sophia-Antipolis Méditerranée, Valbonne, France

November 5, 2015

Abstract

We improve the results by Siegelmann & Sontag [1, 2] by providing a novel and parsimonious constructive mapping between Turing Machines and Recurrent Artificial Neural Networks, based on recent developments of Nonlinear Dynamical Automata. The architecture of the resulting R-ANNs is simple and elegant, stemming from its transparent relation with the underlying NDAs. These characteristics yield promise for developments in machine learning methods and symbolic computation with continuous time dynamical systems. A framework is provided to directly program the R-ANNs from Turing Machine descriptions, in absence of network training. At the same time, the network can potentially be trained to perform algorithmic tasks, with exciting possibilities in the integration of approaches akin to Google DeepMind's Neural Turing Machines.

1 Introduction

The present work provides a novel and alternative approach to the one offered by Siegelmann and Sontag [1, 2] of mapping Turing machines to Recurrent Artificial Neural Networks (R-ANNs). Here we employ recent theoretical developments from symbolic dynamics enabling the mapping from Turing Machines to two-dimensional piecewise affine-linear systems evolving on the unit square, i.e. Nonlinear Dynamical Automata (NDA)[3, 4]. With this in place, we are able to map the resulting NDA onto a R-ANN, therefore providing an elegant constructive method to simulate a Turing machine in real time by a first-order R-ANN. There are two main advantages to the proposed approach. The first one is the parsimony and simplicity of the resulting R-ANN architecture in respect to previous approaches. The second one is the transparent relation between the network and its underlying piecewise affine-linear system. These two characteristics open the door to key future developments when considering learning applications (see Google DeepMind's Neural Turing Machines[5] for a relevant example with promising future integration possibilities) – with the exciting possibility of a symbolic read-out of a learned algorithm from the network weights – and when considering extensions of the model to continuous dynamics, which could provide a theoretical basis to query the computational power of more complex neuronal models.

*giovanni.carmantini@gmail.com

2 Methods

In this section we outline a mapping from Turing machines to R-ANNs. Our construction involves two stages. In the first stage a Generalized Shift [3] emulating a Turing Machine is built, and its dynamics encoded on the unit square via a procedure called Gödelization, defining a piecewise-affine linear map on the unit square, i.e. a NDA. In the second stage, the resulting NDA is mapped onto a first-order R-ANN. Next, the theoretical methods employed are discussed in detail.

2.1 Turing Machines

A Turing Machine [6] is a computing device endowed with a doubly-infinite one-dimensional tape (memory support with one symbol capacity at each memory location), a finite state controller and a read-write head that follows the instructions encoded by a δ transition function. At each step of the computation, given the current state and the current symbol read by the read-write head, the machine controller determines via δ the writing of a symbol on the current memory location, a shift of the read-write head to the memory location to the left ($\mathcal{L}$) or to the right ($\mathcal{R}$) of the current one, and the transition to a new state for the next computation step. At a computation step, the content of the tape together with the position of the read-write head and the current controller state define a machine configuration.

More formally, a Turing Machine is a 7-tuple $M_{\mathrm{TM}} = (Q, \mathbf{N}, \mathbf{T}, q_0, \sqcup, F, \delta)$, where Q is a finite set of control states, $\mathbf{N}$ is a finite set of tape symbols containing the blank symbol $\sqcup$, $\mathbf{T} \subset \mathbf{N} \setminus \{\sqcup\}$ is the input alphabet, q_0 is the starting state, $F \subset Q$ is a set of 'halting' states and δ is a partial transition function, determining the dynamics of the machine. In particular, δ is defined as follows:

$$\delta : Q \times \mathbf{N} \to Q \times \mathbf{N} \times \{\mathcal{L}, \mathcal{R}\}. \tag{1}$$

2.2 Dotted sequences and Generalized Shifts

A Turing machine configuration can be described by a bi-infinite dotted sequence on some alphabet $\mathbf{A}$; it can then be defined as:

$$s = \ldots d_{i_{-3}} d_{i_{-2}} d_{i_{-1}} . d_{i_0} d_{i_1} d_{i_2} \ldots, \tag{2}$$

where $l = \ldots d_{i_{-3}} d_{i_{-2}}$ describes the part of the tape on the left of the read-write head, $r = d_{i_0} d_{i_1} d_{i_2} \ldots$ describes the part on its right, $q = d_{i_{-1}}$ describes the current state of the machine controller, and the dot denotes the current position of the read-write head, i.e. the symbol to its right. The central dot splits the tape into two one-sided infinite strings α', β, where α' is the left part of the dotted sequence in reverse order. The first symbol in α represents the current state of the Turing Machine, whereas the first symbol in β represents the symbol currently under the controller's head. The transition function δ can be straightforwardly extended to a function $\hat{\delta}$ operating on dotted sequences, so that $\hat{\delta} : \mathbf{A}^{\mathbb{Z}} \to \mathbf{A}^{\mathbb{Z}}$.

A Generalized Shift acts on dotted sequences, and is defined as a pair $M_{GS} = (\mathbf{A}^{\mathbb{Z}}, \Omega)$, with $\mathbf{A}^{\mathbb{Z}}$ being the space of dotted sequences, $\Omega : \mathbf{A}^{\mathbb{Z}} \to \mathbf{A}^{\mathbb{Z}}$ defined by

$$\Omega(s) = \sigma^{F(s)}(s \oplus G(s)) \tag{3}$$

with

$$F : \mathbf{A}^{\mathbb{Z}} \to \mathbb{Z} \tag{4}$$

$$G : \mathbf{A}^{\mathbb{Z}} \to \mathbf{A}^{e} \tag{5}$$

where σ shifts the symbols to the left or to the right, or does not shift them at all, as determined by the function $F(s)$. In addition, the Generalized Shift can operate a substitution, with $G(s)$ being the function which substitutes a substring of length e in the *Domain of Effect* (DoE) of s with a new substring. Both the shift and the substitution are functions of the content of the *Domain of Dependence* (DoD), a substring of s of length ℓ.

A Turing Machine can be emulated by a Generalized Shift with DoD = DoE = $d_{i_{-2}} d_{i_{-1}} . d_{i_0}$ and the functions F, G appropriately chosen such that $\Omega(s) = \hat{\delta}(s)$ for all s (see [7] for a detailed exposition).

2.3 Gödel codes

Gödel codes (or Gödelizations) [8] map strings to numbers and, in particular, allow the mapping of the space of one-sided infinite sequences to the real interval $[0, 1]$. Let $\mathbf{A}^{\mathbb{N}}$ be the space of one-sided infinite sequences over an alphabet $\mathbf{A}$, s be an element of $\mathbf{A}^{\mathbb{N}}$, r_k the k-th symbol in s, $\gamma : \mathbf{A} \to \mathbb{N}$ a one-to-one function associating each symbol in the alphabet $\mathbf{A}$ to a natural number, and g the number of symbols in $\mathbf{A}$. Then a Gödelization is a mapping ψ from $\mathbf{A}^{\mathbb{N}}$ to $[0, 1] \subset \mathbb{R}$ defined as:

$$\psi(s) := \sum_{k=1}^{\infty} \gamma(r_k) g^{-k}. \tag{6}$$

Conveniently, Gödelization can be employed on a Turing machine configuration, represented as a dotted sequence $\alpha.\beta \in \mathbf{A}^{\mathbb{Z}}$. The Gödel encoding ψ_x and ψ_y of α' and β define a representation of s $(\psi_x(\alpha'), \psi_y(\beta))$ known as symbol plane or symbologram representation, which is contained in the unit square $[0, 1]^2 \subset \mathbb{R}^2$. The choice of encoding ψ_x and ψ_y to use on the machine configurations is arbitrary. Therefore, to enable the construction of parsimonious Nonlinear Dynamical Automata our encoding will assume that β always contains tape symbols only, and that the first symbol of α' is always a state symbol, the rest being tape symbols only. Based on these assumptions, the particular encoding is defined as:

$$\begin{aligned} \psi_x(\alpha') &= \gamma_q(a_1) n_q^{-1} + \sum_{k=1}^{\infty} \gamma_s(a_{k+1}) n_s^{-k} n_q^{-1}, \\ \psi_y(\beta) &= \sum_{k=1}^{\infty} \gamma_s(b_k) n_s^{-k}, \end{aligned} \tag{7}$$

with $n_q = |Q|$, i.e. the number of states in the Turing Machine, $n_s = |\mathbf{N}|$, i.e. the number of tape symbols in the Turing Machine, γ_q and γ_s enumerating Q and $\mathbf{N}$ respectively, and with a_k and b_k being the k-th symbol in α' and β respectively.

2.3.1 Encoded Generalized Shift and affine-linear transformations

The substitution and shift operated by a Generalized Shift on a dotted sequence $s = \alpha.\beta$ can be represented as an affine-linear transformation on $(\psi_x(\alpha'), \psi_y(\beta))$, i.e. the symbologram representation of s. In particular, a substitution and shift on a dotted sequence can be broken down into substitutions and shifts on its one-sided components. In the following, we will show how substitutions and shifts on a one-sided infinite sequence can be represented as affine-linear transformations on its Gödelization. These results will be useful in showing how the symbologram representation of a Generalized Shift leads to a piecewise affine-linear map on a rectangular partition of the unit square.

Let $s = d_1 d_2 d_3 \ldots$ be a one-side infinite sequence on some alphabet $\mathbf{A}$. Substituting the n-th symbol in s with $\hat{d}_n$ yields $\hat{s} = d_1 \ldots d_{n-1} \hat{d}_n d_{n+1} \ldots$, so that

$$\begin{aligned}\psi(s) &= \gamma(d_1)g^{-1} + \ldots \gamma(d_{n-1})g^{-(n-1)} + \gamma(d_n)g^{-n} + \gamma(d_{n+1})g^{n+1} + \ldots, \\ \psi(\hat{s}) &= \gamma(d_1)g^{-1} + \ldots \gamma(d_{n-1})g^{-(n-1)} + \gamma(\hat{d}_n)g^{-n} + \gamma(d_{n+1})g^{n+1} + \ldots, \\ &= \psi(s) - \gamma(d_n)g^{-n} + \gamma(\hat{d}_n)g^{-n}.\end{aligned}$$

As the previous example illustrates, Gödelizing a sequence resulting from a symbol substitution is equivalent to applying an affine-linear transformation on the original Gödelized sequence. In particular, the parameters of the affine-linear transformation only depend on the position and identities of the symbols involved in the substitution. Shifting s to the left by removing its first symbol or shifting it to the right by adding a new one yields respectively $s_l = d_2 d_3 d_4 \ldots$ and $s_r = b\, d_1 d_2 d_3 d_4 \ldots$, where b is the newly added symbol. In this case

$$\begin{aligned}\psi(s_l) &= \gamma(d_2)g^{-1} + \gamma(d_3)g^{-2} + \gamma(d_4)g^{-3} + \ldots \\ &= g\psi(s) - \gamma(d_1),\end{aligned}$$

and

$$\begin{aligned}\psi(s_r) &= \gamma(b)g^{-1} + \gamma(d_1)g^{-2} + \gamma(d_2)g^{-3} + \gamma(d_3)g^{-4} + \ldots \\ &= g^{-1}\psi(s) + \gamma(b)g^{-1}.\end{aligned}$$

Again, the resulting Gödelized shifted sequence can be obtained by applying an affine-linear transformation to the original Gödelized sequence.

2.4 Nonlinear Dynamical Automata

A Nonlinear Dynamical Automaton (NDA) is a triple $M_{NDA} = (X, P, \Phi)$, with P being a rectangular partition of the unit square, that is

$$P = \{D^{i,j} \subset X |\ 1 \leq i \leq m,\ 1 \leq j \leq n,\ m, n \in \mathbb{N}\}, \tag{8}$$

so that each cell $D^{i,j}$ is defined as the cartesian product $I_i \times J_j$, with $I_i, J_j \subset [0,1]$ being real intervals for each bi-index (i,j), $D^{i,j} \cap D^{k,l} = \emptyset$ if $(i,j) \neq (k,l)$, and $\bigcup_{i,j} D^{i,j} = X$.
The couple (X, Φ) is a time-discrete dynamical system with phase space $X = [0,1]^2 \subset \mathbb{R}^2$ (i.e. the unit square) and with flow $\Phi : X \to X$, a piecewise affine-linear map such that $\Phi_{|D^{i,j}} := \Phi^{i,j}$. Specifically, $\Phi^{i,j}$ takes the following form:

$$\Phi^{i,j}(\mathbf{x}) = \begin{pmatrix} a_x^{i,j} \\ a_y^{i,j} \end{pmatrix} + \begin{pmatrix} \lambda_x^{i,j} & 0 \\ 0 & \lambda_y^{i,j} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}. \tag{9}$$

The piecewise affine-linear map Φ also requires a switching rule $\Theta(x,y) \in [\![1,m]\!] \times [\![1,n]\!]$ to select the appropriate branch, and thus the appropriate dynamics, as a function of the current state. That is, $\Phi(x,y) = \Phi^{i,j}(x,y) \iff \Theta(x,y) = (i,j)$.

Each cell $D^{i,j}$ of the partition P of the unit square can be seen as comprising all the Gödelized dotted sequences that contain the same symbols in the Domain of Dependence. That is, for a Generalized Shift simulating a Turing Machine, the first two symbols in α' and the first symbol in β.
The unit square is thus partitioned in a number of I intervals equal to $m = n_q n_s$, and one of J intervals equal to $n = n_s$, with n_q being the number of states in Q and n_s the number of symbols

in $\mathbf{N}$, for a total of $n_q n_s^2$ cells. As each cell corresponds to a different Domain of Dependence of the underlying Generalized Shift in symbolic space, it is associated with a different affine-linear transformation representing the action of a substitution and shift in vector space. The transformation parameters $(a_x^{i,j}, a_y^{i,j})$ and $(\lambda_x^{i,j}, \lambda_y^{i,j})$ can be derived using the methods outlined in subsubsection 2.3.1.
Thus, a Turing Machine can be represented as a Nonlinear Dynamical Automaton by means of its Gödelized Generalized Shift representation.

3 NDAs to R-ANNs

The aim of the second stage of our methodology is to map the orbits of the NDA (i.e. $\Phi^{i,j}(x, y)$) to orbits of the R-ANN, which we will denote by $\zeta^{i,j}(x, y)$.

Let $\rho(\cdot)$ denote the proposed map. Its role is to encode the affine-linear dynamics at each $\Phi^{i,j}$ branch in the architecture and weights of the network, and emulate the overall dynamics Φ by suitably activating certain neural units within the R-ANN given the switching rule Θ. Therefore, we generically define the proposed map as follows:

$$\zeta = \rho(\mathcal{I}, \mathcal{A}, \Phi, \Theta), \tag{10}$$

where $\mathcal{I}$ is the identity matrix mapping (identically) the initial conditions of the NDA to the R-ANN and $\mathcal{A}$ is the adjacency matrix specifying the network architecture and weights, which will be explained in subsequent sections. In addition, ρ defines different neural dynamics for each type of the neural units, that is, $\zeta = (\zeta_1, \zeta_2, \zeta_3)$ corresponding to MCL, BSL and LTL, respectively (see below for the definitions of these acronyms). The details of the R-ANN architecture and its dynamics are subsequently discussed.

3.1 Network architecture and neural dynamics

The proposed map, ρ, attempts to mirror the affine-linear dynamics (given by Equation 9) of an NDA on the partitioned unit square (see Equation 8) by endowing the R-ANN with a structure capturing the characteristic features of a piecewise-affine linear system, i.e. a state, a switching rule and a set of transformations.
To achieve this, we propose a network architecture with three layers, namely a Machine Configuration Layer (MCL) encoding the state, a Branch Selection Layer (BSL) implementing the switching rule and a Linear Transformation Layer (LTL), as depicted in Figure 1.
The neural units within the various layers make use of either the Heaviside (H) or the Ramp (R) activation functions defined as follows:

$$H(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \tag{11}$$

$$R(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}. \tag{12}$$

Since Φ is a two-dimensional map, this suggests only two neural units (c_x, c_y) in the MCL layer encoding its state at every step. A set of BSL units functionally acts as a switching system that determines in which cell $D^{i,j}$ the current Turing machine configuration belongs to and then triggers the specific LTL unit emulating the application of an affine-linear transformation $\Phi^{i,j}$ on the current state of the system. The result of the transformation is then fed back to the MCL for the next iteration. On the symbolic level, one iteration of the emulated NDA corresponds to a tape and state update of the underlying Turing machine, which can be read out by decoding the activation of the MCL neurons.

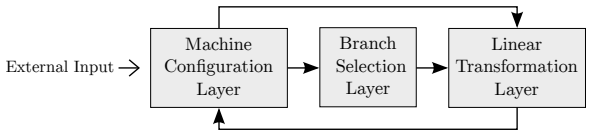
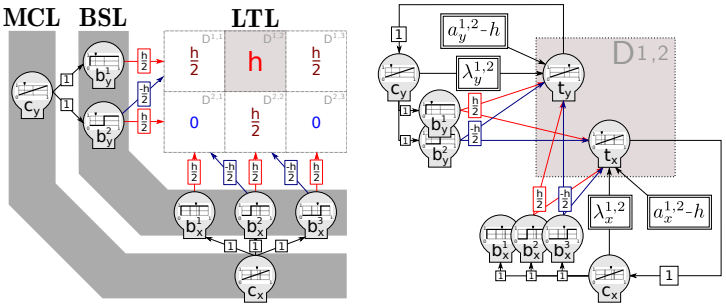


Figure 1. Connectivity between neural layers within the network.



(a) Branch Selection Layer (b) Complete branch connection layout

Figure 2. Detailed feedforward connectivity and weights for a neural network simulating a NDA with only 6 branches.

3.1.1 Machine Configuration Layer

The role of the MCL is to store the current Gödelized configuration of the simulated Turing Machine at each computation step, and to synaptically transmit it to the BSL and LTL layers. The layer comprises two neural units (c_x and c_y), as needed to store the Gödelized dotted sequence representing a Turing Machine configuration (see Equation 7).

The R-ANNs is thus initialized by activating this layer, given the NDA initial conditions $(\psi_x(\alpha'), \psi_x(\beta))$ which are identically transformed via $\mathcal{I}$ by the map $\rho(\cdot)$ as follows:

$$(c_x, c_y) = (\psi_x(\alpha'), \psi_x(\beta)) \equiv \zeta_1 = \rho(\mathcal{I}, \cdot, \cdot, \cdot)_{|(\psi_x(\alpha'), \psi_x(\beta))} \tag{13}$$

At each iteration, the units in this layer receive input from the LTL units, and are activated via the ramp activation function (Equation 12); in other words $\zeta_1 \equiv (c_x, c_y) = (R(\sum_i t_x^i), R(\sum_j t_y^j))$. Finally, the MCL synaptically projects onto the BSL and LTL (refer to Figure 2 for details of the connectivity).

3.1.2 Branch Selection Layer

The BSL embodies the switching rule $\Theta(x, y)$ and coordinates the dynamic switching between LTL units. In particular, if at the current step the MCL activation is $(c_x, c_y) \in D^{i,j} = I_i \times J_j$, with $I_i = [\xi_i, \xi_{i+1})$ being the i-th interval on the x-axis and $J_j = [\eta_j, \eta_{j+1})$ being the j-th interval on the y-axis, the BSL units activate only the $(t_x^{i,j}, t_y^{i,j})$ units in the LTL. In this way, only one couple of LTL units is active at each step. The switching rule is mapped by $\rho(\cdot)$ as

follows:

$$\zeta_2(x,y) = \rho(\cdot,\cdot,\cdot,\Theta(x,y) = (i,j)). \tag{14}$$

The BSL is composed of two groups of Heaviside (Equation 11) units, implementing respectively the x and the y component of the switching rule of the underlying piecewise affine-linear system, namely: i) the b_x group receives input with weight 1 from the c_x unit of the MCL layer, and comprises $n_q n_s$ units (i.e. $b_x^i, 1 \leq i \leq n_q n_s$); ii) the b_y group receives input with weight 1 from c_y and comprises n_s units (i.e. $b_y^j, 1 \leq i \leq n_s$). The activation of the two groups of units is defined as:

$$\begin{aligned} b_x^i &= H(c_x - \xi^i) \qquad \text{with} \qquad \xi^i = \min(I_i), \\ b_y^j &= H(c_y - \eta^j) \qquad \text{with} \qquad \eta^j = \min(J_i). \end{aligned} \tag{15}$$

Each b_x^i and b_y^j BSL unit has an activation threshold, defined as the left boundary of the I_i and J_j intervals, respectively, and implemented as input from an always-active bias unit (with weight $-\xi^i$ for the b_x^i unit and $-\eta^j$ for b_y^j). Therefore, an activation of (c_x, c_y) in the MCL corresponding to a point on the unit square belonging to cell $D^{i,j}$, would trigger active all units b_x^k with $k \leq i$. The same would occur for all neural units b_y^k with $k \leq j$.[1]

Each b_x^i unit establishes synaptic excitatory connections (with weight $\frac{h}{2}$) to all LTL units corresponding to cells $D^{k,i}$ (i.e. $(t_x^{k,i}, t_y^{k,i})$) and inhibitory connections (with weight $\frac{-h}{2}$) to all LTL units corresponding to cells $D^{k,i-1}$ (i.e. $(t_x^{k,i-1}, t_y^{k,i-1})$), with $k = 1, \ldots, n_s$; for a graphical representation see Figure 2. Similarly, each b_y^j unit establishes synaptic excitatory connections to all LTL units corresponding to cells $D^{j,k}$ and inhibitory connections to all LTL units corresponding to cells $D^{j-1,k}$, with $k = 1, \ldots, n_q n_s$. Together, the b_x^i and b_y^j units completely counterbalance through their synaptic excitatory connections the natural inhibition (of bias h, which value and definition will be discussed in the following section) of the LTL units corresponding to cell $D^{i,j}$ (i.e. $(t_x^{i,j}, t_y^{i,j})$).

In other words each couple of LTL units $(t_x^{i,i}, t_y^{i,j})$ receives an input of $B_x^i + B_y^j$, defined as follows:

$$\begin{aligned} B_x^i &= b_x^i \frac{h}{2} + b_x^{i+1} \frac{-h}{2}, \\ B_y^j &= b_y^j \frac{h}{2} + b_y^{j+1} \frac{-h}{2}, \end{aligned} \tag{16}$$

where the input sum

$$B_x^i + B_y^j = \begin{cases} h & \text{if } (c_x, c_y) \in D_{i,j} \\ \frac{h}{2} & \text{if } c_x \in I_i, c_y \notin J_j \quad \text{or} \quad c_x \notin I_i, c_y \in J_j \\ 0 & \text{if } (c_x, c_y) \notin D_{i,j} \end{cases} \tag{17}$$

only triggers the relevant LTL unit if it reaches the value h. That is, if $(c_x, c_y) \in D_{i,j}$ then $B_x^i + B_y^j = h$, and the pair $(t_x^{i,i}, t_y^{i,j})$ is selected by the BSL units. Otherwise $(t_x^{i,i}, t_y^{i,j})$ stays inactive as $B_x^i + B_y^j$ is either equal to $\frac{h}{2}$ or 0, which is not enough to win the LTL pair natural inhibition. An example of this mechanism is shown in Figure 2 , where the LTL units in cell $D^{1,2}$ are activated via mediation of $b_x = \{b_x^1, b_x^2, b_x^3\}$ and $b_y = \{b_y^1, b_y^2\}$. Here, both b_x^3 and b_y^2 are not excited since c_x and c_y, respectively, are not activated enough to drive them towards their threshold. However, b_x^2 excites (with weights $\frac{h}{2}$) the LTL units in cell $D^{2,2}$ and $D^{1,2}$ and inhibits (with weights $\frac{-h}{2}$) the LTL units in cell $D^{2,1}$ and $D^{1,1}$. Equally, b_y^2 excites (with weights

[1] Note that the action of the BSL could be equivalently implemented by interval indicator functions represented as linear combinations of Heaviside functions.

$\frac{h}{2}$) the LTL units in cell $D^{2,1}$, $D^{2,2}$ and $D^{2,3}$ and inhibits (with weights $\frac{-h}{2}$) the LTL units in cells $D^{1,1}$, $D^{1,2}$ and $D^{1,3}$. The b^1_x and b^1_y units excite cells $\{D^{2,1}, D^{1,1}\}$ and $\{D^{1,1}, D^{1,2}, D^{1,3}\}$, respectively, but these do not inhibit any cells (due to boundary conditions).

3.1.3 Linear Transformation Layer

The LTL layer can be functionally divided in sets of two units, where each couple applies two decoupled affine-linear transformations corresponding to one of the branches of the simulated NDA. On the symbolic level, this endows the LTL with the ability to generate an updated machine configuration from the previous one. In the LTL, a branch (i, j) of a NDA, $\Phi^{i,j}(x, y) = (\lambda^{i,j}_x x + a^{i,j}_x, \lambda^{i,j}_y y + a^{i,j}_y)$, is simulated by the LTL units $(t^{i,j}_x, t^{i,j}_y)$. Mathematically, this induces the following mapping:

$$(t^{i,j}_x, t^{i,j}_y) \quad = \quad \zeta^{i,j}_3(x, y) = \rho(\cdot, \cdot, \Phi^{i,j}(x, y), \cdot). \tag{18}$$

The affine-linear transformation is implemented synaptically, and it is only triggered when the BSL units provide enough excitation to enable $(t^{i,j}_x, t^{i,j}_y)$ to cross their threshold value and execute the operation. The read-out of this process corresponds to:

$$\begin{array}{rcl} t^{i,j}_x & = & R(\lambda^{i,j}_x c_x + a^{i,j}_x - h + B^i_x + B^j_y), \\ t^{i,j}_y & = & R(\lambda^{i,j}_y c_y + a^{i,j}_y - h + B^i_x + B^j_y). \end{array} \tag{19}$$

A strong inhibition bias h (implemented as a synaptic projection from a bias unit) plays a key role in rendering the LTL units inactive in absence of sufficient excitation. The bias value is defined as follows

$$-\frac{h}{2} \leq -\max_{i,j,k}(a^{i,j}_k + \lambda^{i,j}_k) \quad \text{with} \quad k = \{x, y\}. \tag{20}$$

Hence, each of the BSL inputs B^i_x and B^i_y contributes respectively to half of the necessary excitation ($\frac{h}{2}$) needed to counterbalance the LTL's natural inhibition (refer to Equation 16 and Equation 17).

The LTL units receive input from the two CSL units (c_x, c_y), with synaptic weights of $(\lambda^{i,j}_x, \lambda^{i,j}_y)$, and they are also endowed with an intrinsic constant LTL neural dynamics $(a^{i,j}_x, a^{i,j}_y)$. If the input from the BSL layer is enough for these neurons to cross the threshold mediated by the Ramp activation function, the desired affine-linear transformation is applied. The read-out is an updated encoded Turing machine configuration, which is then synaptically fed back to the CSL units (c_x, c_y), ready for the next iteration (or next Turing machine computation step on the symbolic level).

3.1.4 NDA-simulating first order R-ANN

The NDA simulation (and thus Turing machine simulation) by the R-ANN is achieved by a combination of synaptic and neural computation among the three neural types (MCL, BSL, and LTL) and with a total of

$$n_{\text{units}} = \underbrace{2}_{\text{MCL}} + \underbrace{n_s + n_s n_q}_{\text{BSL}} + \underbrace{2n_s^2 n_q}_{\text{LTL}} + \underbrace{1}_{\text{bias unit}} \tag{21}$$

neural units, where n_q and n_s are the number of states and the number of symbols in the Turing Machine to be simulated, respectively. These units are connected as specified by an adjacency

matrix $\mathcal{A}$ of size $n_{\text{units}} \times n_{\text{units}}$, following the connectivity pattern described in Figure 1 and with synaptic weights as entries from the set

$$\{0, 1, \frac{h}{2}, \frac{-h}{2}\} \cup \{a_k^{i,j} - h \mid i = 1, \ldots, n_q n_s \ , \ j = 1, \ldots, n_s \ , \ k = x, y\},$$

the second component being the set of biases.

An important modelling issue to consider is that of the halting conditions for the ANN, i.e. when to consider the computation completed. In the original formulation of the Generalized Shift, there is no explicit definition of halting condition. As our ANN model is based on this formulation, a deliberate choice has to be made in its implementation. Two choices seem to be the most reasonable. The first one involves the presence of an external controller halting the computation when some conditions are met, i.e. an *homunculus* [4]. The second one is the implementation of a fixed point condition, intrinsic to the dynamical system, representing a TM halting state as an Identity branch on the NDA. In this way a halting configuration will result in a fixed point on the NDA, and thus on the R-ANN. In other words, the network's computation is considered completed if and only if

$$\zeta_1(x', y') = (x', y'). \tag{22}$$

In the present study we decided to use a fixed point halting condition, but the use of a *homunculus* would likely be more appropriate in other contexts such as interactive computation [9, 10, 11] or cognitive modelling, where different kinds of fixed points are required in order to describe sequential decision problems [12], such as linguistic garden paths [4, 10].

The implementation of the R-ANN defined like so simulates a NDA in real-time and, thus, it simulates a Turing Machine in real time. More formally, it can be shown that under the map $\rho(\cdot)$ the commutativity property $\zeta \circ \rho = \rho \circ \Phi$ is satisfied, which extends the previously demonstrated commutativity property between Turing machines and NDAs [9, 13, 14].

4 Discussion

In this study we described a novel approach to the mapping of Turing Machines to first-order R-ANNs. Interestingly, R-ANNs can be constructed to simulate any piecewise affine-linear system on a rectangular partition of the n-dimensional hypercube by extending the methods discussed

The proposed mapping allows the construction, given any Turing Machine, of a R-ANN simulating it in real time. As an example of the parsimony we claim, a Universal Turing Machine can be simulated with a fraction of the units than previous approaches allowed for: the proposed mapping solution derives a R-ANN that can simulate Minsky's 7-states 4-symbols UTM [15] in real-time with 259 units (as per Equation 21), approximately 1/3 of the 886 units needed in the solution proposed by Siegelmann and Sontag [1], and with a much simpler architecture.

In future work we plan to overcome some of the issues posed by the mapping and parts of its underlying theory, especially in relation to learning applications. Key issues to overcome are the missing end-to-end differentiability, and the need for a de-coupling of states and data in the encoding. A future development would see the integration of methods of data access and manipulation akin to that in Google DeepMind's Neural Turing Machines [5]. A parallel direction of future work would see the mapping of Turing machines to continuous-time dynamical systems (an example with polynomial systems is provided in [16]). In particular, heteroclinic dynamics [12, 13, 17, 18] – with machine configurations seen as metastable states of a dynamical system – and slow-fast dynamics [19, 20] are promising new directions of research.

References

[1] H. T. Siegelmann and E. D. Sontag, "On the computational power of neural nets," *Journal of computer and system sciences*, vol. 50, no. 1, pp. 132–150, 1995.

[2] H. T. Siegelmann and E. D. Sontag, "Turing computability with neural nets," *Appl. Math. Lett*, vol. 4, no. 6, pp. 77–80, 1991.

[3] C. Moore, "Unpredictability and undecidability in dynamical systems," *Physical Review Letters*, vol. 64, no. 20, p. 2354, 1990.

[4] P. beim Graben, B. Jurish, D. Saddy, and S. Frisch, "Language processing by dynamical systems," *International Journal of Bifurcation and Chaos*, vol. 14, no. 02, pp. 599–621, 2004.

[5] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014.

[6] A. M. Turing, "On computable numbers, with an application to the entscheidungsproblem," *Proc. London Math. Soc*, vol. 42, 1937.

[7] C. Moore, "Generalized shifts: unpredictability and undecidability in dynamical systems," *Nonlinearity*, vol. 4, no. 2, p. 199, 1991.

[8] K. Gödel, "Über formal unentscheidbare sätze der *principia mathematica* und verwandter systeme i," *Monatshefte für Mathematik und Physik*, vol. 38, pp. 173 – 198, 1931.

[9] P. beim Graben, "Quantum Representation Theory for Nonlinear Dynamical Automata," in *Advances in Cognitive Neurodynamics ICCN 2007*, pp. 469–473, Springer, 2008.

[10] P. beim Graben, S. Gerth, and S. Vasishth, "Towards dynamical system models of language-related brain potentials," *Cognitive neurodynamics*, vol. 2, no. 3, pp. 229–255, 2008.

[11] P. Wegner, "Interactive foundations of computing," *Theoretical Computer Science*, vol. 192, pp. 315 – 351, 1998.

[12] M. I. Rabinovich, R. Huerta, P. Varona, and V. S. Afraimovich, "Transient cognitive dynamics, metastability, and decision making," *PLoS Computational Biology*, vol. 4, no. 5, p. e1000072, 2008.

[13] P. beim Graben and R. Potthast, "Inverse problems in dynamic cognitive modeling," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 19, no. 1, p. 015103, 2009.

[14] P. beim Graben and R. Potthast, "Universal neural field computation," in *Neural Fields*, pp. 299–318, Springer, 2014.

[15] M. Minsky, "Size and structure of universal turing machines using tag systems," in *Recursive Function Theory: Proceedings, Symposium in Pure Mathematics*, vol. 5, pp. 229–238, 1962.

[16] D. S. Graça, M. L. Campagnolo, and J. Buescu, "Computability with polynomial differential equations," *Advances in Applied Mathematics*, vol. 40, no. 3, pp. 330–349, 2008.

[17] I. Tsuda, "Toward an interpretation of dynamic neural activity in terms of chaotic dynamical systems," *Behavioral and Brain Sciences*, vol. 24, pp. 793 – 810, 2001.

[18] M. Krupa, "Robust heteroclinic cycles," *Journal of Nonlinear Science*, vol. 7, no. 2, pp. 129–176, 1997.

[19] M. Desroches, M. Krupa, and S. Rodrigues, "Inflection, canards and excitability threshold in neuronal models," *Journal of mathematical biology*, vol. 67, no. 4, pp. 989–1017, 2013.

[20] M. Desroches, A. Guillamon, R. Prohens, E. Ponce, S. Rodrigues, and A. E. Teruel, "Canards, folded nodes and mixed-mode oscillations in piecewise-linear slow-fast systems," *SIAM Review*, vol. in press, 2015.

A modular architecture for transparent computation in Recurrent Neural Networks

Giovanni S. Carmantini[a,*], Peter beim Graben[b], Mathieu Desroches[c], Serafim Rodrigues[a]

[a] *School of Computing and Mathematics, Plymouth University, Plymouth, United Kingdom*
[b] *Bernstein Center for Computational Neuroscience Berlin, Humboldt-Universität zu Berlin, Berlin, Germany*
[c] *Inria Sophia-Antipolis Méditerranée, Valbonne, France*

Abstract

Computation is classically studied in terms of automata, formal languages and algorithms; yet, the relation between neural dynamics and symbolic representations and operations is still unclear in traditional eliminative connectionism. Therefore, we suggest a unique perspective on this central issue, to which we would like to refer as to transparent connectionism, by proposing accounts of how symbolic computation can be implemented in neural substrates. In this study we first introduce a new model of dynamics on a symbolic space, the versatile shift, showing that it supports the real-time simulation of a range of automata. We then show that the Gödelization of versatile shifts defines nonlinear dynamical automata, dynamical systems evolving on a vectorial space. Finally, we present a mapping between nonlinear dynamical automata and recurrent artificial neural networks. The mapping defines an architecture characterized by its granular modularity, where data, symbolic operations and their control are not only distinguishable in activation space, but also spatially localizable in the network itself, while maintaining a distributed encoding of symbolic representations. The resulting networks simulate automata in real-time and are programmed directly, in absence of network training. To discuss the unique characteristics of the architecture and their consequences, we present two examples: i) the

*Corresponding author
Email address: giovanni.carmantini@gmail.com (Giovanni S. Carmantini)

design of a Central Pattern Generator from a finite-state locomotive controller, and ii) the creation of a network simulating a system of interactive automata that supports the parsing of garden-path sentences as investigated in psycholinguistics experiments.

Keywords: Automata Theory, Recurrent Artificial Neural Networks, Representation Theory, Nonlinear Dynamical Automata, Neural Symbolic Computation, Versatile Shift

1. Introduction

The relation between symbolic computation and neural dynamics is one of the most pertinent problems in computational neuroscience, artificial intelligence, and cognitive science. On the one hand, symbolic computation is generically codified in terms of production systems, formal languages, algorithms and automata (Hopcroft and Ullman, 1979). On the other hand, neural dynamics in artificial neural networks (ANN) is described by nonlinear evolution laws (Hertz et al., 1991). Approaches to connect these different realms of research go back to the seminal paper of McCulloch and Pitts (1943) on networks of idealized two-state neurons that behave as logic gates. Furthermore, fundamental work by Kleene (1956) and Minsky (1967) demonstrated the equivalence between such networks and finite-state automata, and thus digital computers (which are essentially large-scale networks of logic gates). Later examples for connectionist modeling of symbolic computation are the speech perception and production models *TRACE* by McClelland and Elman (1986) and *NETtalk* by Sejnowski and Rosenberg (1987). A further important step was achieved by Elman when introducing simple recurrent networks (SRN) as prediction devices for letters in words (Elman, 1990) and syntactic categories in sentences (Elman, 1995). SRN found a number of successful applications in linguistics and cognitive science (Tabor et al., 1997; Christiansen and Chater, 1999; Lawrence et al., 2000; Farkas and Crocker, 2008) where formal grammars have been employed for the generation of training sets. After training, grammatical relations emerged in the connectivity and activation patterns of the network's hidden layer which could be examined through clustering and principal component analysis (PCA).

A key problem of this and similar approaches based on *eliminative connectionism* (Blutner, 2011), a theoretical stance aiming at the elimination of symbolic representations in connectionist models, is that the emerging rep-

resentations, while comparable in a metric space through empirical methods such as clustering or PCA, do not allow inferences about the syntactic or structural relationships of the symbolic training data. This is even more the case with contemporary deep-learning (Li, 2014; Bengio et al., 2013), and reservoir computing approaches featuring large networks of randomly and recurrently connected nonlinear units (Dominey, 1995; Jaeger, 2001; Maass et al., 2002; Steil, 2004). For that reason, another branch of research, which we may call *transparent connectionism*, has been developed in the framework of vector symbolic architectures (VSA) (Mizraji, 1989; Smolensky, 1990; Smolensky and Legendre, 2006a,b; Gayler, 2006; Gayler et al., 2010; beim Graben and Potthast, 2009). Here, one explicitly starts with the symbolic data structures and processes, which are first decomposed into so-called filler-role bindings and then used to create vectorial images through tensor product representations (Smolensky, 1990; beim Graben and Potthast, 2009). These serve as training patterns for subsequent connectionist modeling. In contrast to eliminative connectionism where representations that emerge during training are to a great extent opaque, representations in VSAs are completely transparent as they can be resolved in each step of the encoding procedure. Depending on the structure of the chosen vector space one arrives at different kinds of integrated connectionist/symbolic architectures (ICS) (Smolensky, 1990; Smolensky and Legendre, 2006a,b): Gödelizations for one-dimensional representations in the field of real numbers, proper vectorial representations for finite-dimensional vector spaces, and functional representations for infinite-dimensional vector spaces (beim Graben and Potthast, 2009). Importantly, Siegelmann and Sontag (1991, 1995) used a combination of Gödelization and localist finite-dimensional representation to prove that Recursive ANNs (R-ANN) with rational weights and ramp activation functions can simulate any n-tape ($n \geq 2$) stack machine – or, equivalently, any Turing machine (TM) and any partial recursive function – when endowed with a specific localist architecture. Moreover, Siegelmann and Sontag showed that a R-ANN consisting of 886 units can simulate a universal Turing machine (UTM). Recent work by Cabessa (Cabessa and Siegelmann, 2012; Cabessa and Villa, 2012, 2013) extends these results on R-ANNs to the realm of interactive computation (Wegner, 1998), a framework studying systems that can interact with the environment throughout their computation (as opposed to the framework of classical computation, where the interaction is limited to the input-output exchange), proving that R-ANNs are equivalent in power to interactive TMs.

Very-large-scale and reservoir-like neural network approaches can also rely on VSA as a key ingredient, as in the *neural engineering* framework (Eliasmith et al., 2012; Stewart et al., 2014), which employs semantic pointers for addressing symbolic representations in activation space, and recent work at the interface between *reservoir computing* and connectionist/symbolic approaches (Hinaut and Dominey, 2013; Hinaut et al., 2014)

In contrast, the present work focuses on parsimonious VSA implementations, building upon the seminal results from Siegelmann and Sontag (1991, 1995), and work from Moore (1990, 1991) who has shown that nonlinear dynamical automata (NDA), piecewise-affine linear dynamical systems on the unit square, can simulate the dynamics of any TM in real-time[1] when the machine is represented as a generalized shift (GS) on dotted sequences. In this work we first extend Moore's results by showing that NDA can support the real-time simulation of a range of models of computation, including but not limited to Turing Machines (of course, TMs can simulate any other model of computation of lesser or equal power, but not necessarily in real-time; see Section 2.1.1 for a discussion). We achieve this by relaxing the definition of GS, which leads to a novel and more expressive shift map, the versatile shift (VS) which enables the parsimonious and real-time emulation of symbolic computation in a range of models. We then show that VS dynamics can be mapped to NDA dynamics on the unit square through Gödelization. Finally, we present a mapping between VS and R-ANNs through NDA (extending preliminary results shown in Carmantini et al., 2015).

Symbolic models of computation distinguish between data, operations on data and the control of these operations. For example, automata implement a set of symbolic operations and its control through a look-up table (the transition function), and the data as a string encoding the so-called *configuration* of the automaton. In grammars and term rewriting systems, operations are instead defined as a set of substitution/rewriting rules on some symbolic string, where the application of these rules is controlled by a set of conditions. NDA can perform symbolic computation on a vectorial space while preserving, in their formulation, the division between data, operations on data, and their control. Basing our construction on NDA, we derive an architecture that also preserves this division, thus obtaining networks that are

[1] In a real-time simulation, a single computation step in the original model is mapped to a single computation step in the model simulating it.

transparent, modular and parsimonious. Importantly, the operations embedded within the architecture we propose herein are not only distinguishable in activation space, but are also spatially localized, while still relying on a distributed representation of the symbolic data. The granular modularity of the architecture brought about by its relation with NDA differentiates our approach from previous work, and has important consequences for the constructive mapping of interactive automata networks (IANs) to R-ANNs, and for the possibility of correlational studies with electrophysiological data, which we will discuss in subsequent Sections.

We illustrate our approach by means of two examples. As a first example, we construct a central pattern generator (CPG) from a finite-state automaton for gait patterns of quadruped animals (Grillner and Zangger, 1975; Collins and Richmond, 1994; Golubitsky et al., 1999). The neuronal sequential activations by CPGs are usually modeled through networks of coupled nonlinear oscillators that undergo symmetry-breaking bifurcations under changes in their driving input (Golubitsky et al., 1999, 1998; Schöner et al., 1990; Collins and Richmond, 1994). We show that our construction, although symbolically inspired, allows the investigation of similar bifurcation scenarios. Additionally, the results of these example are relevant to the design of CPGs for the control of robotic locomotion (Ijspeert, 2008). As a second example, we show how our approach is ideally suited to tackle the mapping of interactive machines to neural networks, because of the separation in the network architecture of data, transformations and their control. This makes it straightforward to construct R-ANNs simulating networks of automata that e.g. share states, are organized in complex hierarchies, or are bound by interactions of conditions in the application of symbolic transformations. We demonstrate this by constructing an interactive automata network (IAN) that implements a diagnosis and repair parser for syntactic language processing (Lewis, 1998) and by subsequently mapping it to a R-ANN performing the same computation. We are then able to derive vectorial observables from the network; specifically, we compute synthetic event-related brain potentials (synth-ERPs, Barrès et al., 2013) and discuss their relation with event-related potentials as measured in experiments involving garden-path sentences (Frisch et al., 2004).

Abbreviation	Extended name
ANN	Artificial neural network
BSL	Branch selection layer
CFG	Context-free grammar
CL	Configuration layer
CPG	Central pattern generator
EEG	Electroencephalography
ERP	Event-related brain potentials
FSM	Finite-state machine
GS	Generalized shift
LFP	Local field potentials
LTL	Linear transformation layer
MCL	Machine configuration layer
NDA	Nonlinear dynamical automaton
PCA	Principal component analysis
PDA	Push-down automaton
R-ANN	Recurrent artificial neural network
SRN	Simple recurrent network
synth-ERP	Synthetic event-related brain potential
TDR	Top-down recognizer
TM	Turing machine
UTM	Universal Turing machine
VS	Versatile shift
VSA	Vector symbolic architecture

Table 1: **List of abbreviations used in this paper.**

2. Methods

The present Section outlines our general method which allows the mapping of a range of models of computation to R-ANNs. In Figure 1 we summarize the complete mapping procedure to accompany its exposition. Our construction is a two-step process. We first define a Versatile shift (a generalization of the shift map introduced in Moore, 1990) that emulates some model of computation, and we subsequently encode its dynamics on the unit square via Gödelization, obtaining a two-dimensional piecewise affine-linear map on the unit square, i.e. a NDA. As a second step, the NDA is mapped onto a first-order R-ANN, which is endowed with an architecture that captures the NDA's three key components: i) a state, encoding the symbolic data of the model of computation; ii) a set of affine-linear transformations, encoding its operations on data; iii) a switching rule that selects the relevant affine-linear transformation to apply given the state, thus implementing the control of the symbolic operations.

Next, the theoretical methods employed are discussed in detail. In the presentation of various objects from Formal Language Theory and Automata Theory, we essentially follow the well-established definitions in Hopcroft and Ullman (1979), and in Sipser (2006).

2.1. Elements of Symbolic Computation

A symbol is meant to be a distinguished element from a finite set $\mathbf{A}$, which we call an *alphabet*. Symbols can be concatenated, i.e. for $a, b \in \mathbf{A}$, $ab \equiv (a, b) \in \mathbf{A}^2$. A sequence of symbols $w \in \mathbf{A}^n$ is called a word of length n, denoted $n = |w|$. The set of words of all possible lengths w of finite length $|w| \geq 0$ is denoted $\mathbf{A}^*$ (for $|w| = 0$, $w = \epsilon$ denotes the "empty word").

2.1.1. From Generalized to Versatile Shifts

The theory of symbolic dynamics (Lind and Marcus, 1995) is a tool to study dynamical systems based on the discretization of time and space in order to interpret trajectories in a vectorial space as discrete sequences of infinite strings of symbols. Importantly, its theoretical apparatus can also be used to do the opposite, mapping sequences of strings to a vectorial space. We start by redefining a representation for strings of symbols, the dotted sequence.

According to Moore (1990, 1991), a *dotted sequence* $s \in \mathbf{A}^{\mathbb{Z}}$ on an alphabet $\mathbf{A}$ is a two-sided infinite sequence of symbols "$s = \ldots\ d_{-2}\ d_{-1}\ .\ d_0\ d_1\ d_2\ \ldots$"

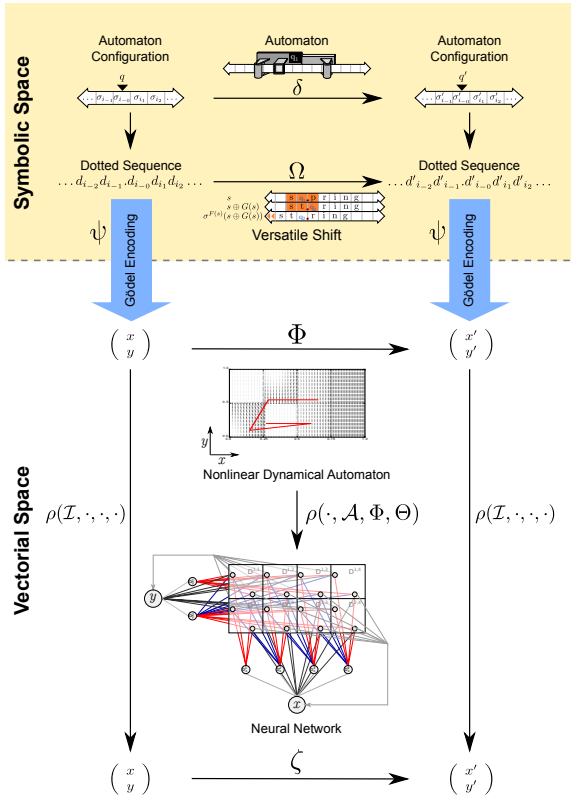


Figure 1: **An automaton is mapped to a recurrent artificial neural network (R-ANN).** The representation of machine configurations as dotted sequences allows for the mapping of the machine transition function to the action of a Ω versatile shift (VS) map upon said sequences, simulating the computation performed by the automaton. A Gödel encoding ψ acts as a bridge between the Symbolic and the Vectorial representation of the automaton's dynamics, and enables the representation of Ω as an affine-linear map Φ by a nonlinear dynamical automaton (NDA). Finally, a map ρ generates a R-ANN, with a specific network architecture and internal dynamics ζ that operates on the same Vectorial space as Φ, where the NDA states are identically mapped through $\rho(I,\cdot,\cdot,\cdot)$ to the activation of a specialized layer in the R-ANN.

where $d_i \in \mathbf{A}$, for all indices $i \in \mathbb{Z}$. Here, the dot "." is simply used as a mnemonic sign, indicating that the index 0 is to its right. A shift space $M_S = (\mathbf{A}^{\mathbb{Z}}, \sigma)$ is then given by a shift map $\sigma : \mathbf{A}^{\mathbb{Z}} \to \mathbf{A}^{\mathbb{Z}}$ (Lind and Marcus, 1995), such that $\sigma(s)_i = (s)_{i+1}$, i.e. σ shifts all symbols in s one place to the left (or, equivalently, shifts the dot one place to the right). Similarly, it is possible to define an inverse to the shift map, σ^{-1}, shifting all symbols in s one place to the right (or, equivalently, the dot one place to the left).

Notice how shifting the dot in a dotted sequence to the left or the right resembles the movement of the read-write head of a Turing machine on its tape (see section 2.1.2 for more details on Turing machines). In order to fully attain the power of Turing machines, Moore (1990, 1991) endows the shift space M_S with three additional maps

$$\begin{aligned} F &: \mathbf{A}^{\mathbb{Z}} \to \mathbb{Z} \\ \oplus &: \mathbf{A}^{\mathbb{Z}} \times (\mathbf{A} \cup \{\phi\})^{\mathbb{Z}} \to \mathbf{A}^{\mathbb{Z}} \\ G &: \mathbf{A}^{\mathbb{Z}} \to (\mathbf{A} \cup \{\phi\})^{\mathbb{Z}}, \end{aligned} \tag{1}$$

such that their composition $\Omega(s) = \sigma^{F(s)}(s \oplus G(s))$ can fully simulate any Turing machine. The augmented shift space $M_{GS} = (\mathbf{A}^{\mathbb{Z}}, \Omega)$ is called *generalized shift* (GS) if there is an open interval of indices around the dot, called *Domain of Dependence* DoD $= (k_l, k_r)$ $(k_l \leq 0 \leq k_r)$, such that $F(s)$ and $G(s)$ only depend on the content of s within the DoD, $F(s)$ determines a number of left shifts $(F(s) > 0)$, right shifts $(F(s) < 0)$, or no shift at all $(F(s) = 0)$ and $G(s)$ maps the symbols s_i within the DoD onto other symbols g_i, while all symbols outside the DoD are mapped onto an auxiliary symbol ϕ. Finally, the composition operator overwrites all symbols s_i within the DoD through their images g_i under G while not changing s outside the DoD, i.e. $(s \oplus g)_i = s_i$ if $g_i = \phi$, but $(s \oplus g)_i = g_i$ if $g_i \neq \phi$.[2]

According to Moore's proof (Moore, 1990, 1991), any Turing machine can be realized as a GS M_{GS}. Since Turing machines can be programmed to simulate the computation carried out by any model of lower or equal computational power, such as finite-state automata or push-down automata,

[2]In his 1991 paper, Moore actually defines the DoD of a GS as a finite set of integers which need not be consecutive, and introduces a second finite set of integers, the *Domain of Effect* (DoE) to indicate the cells to be rewritten (as a function of the cells in the DoD). Nevertheless, it is always possible, given any GS with arbitrary DoD and DoE, to construct an equivalent GS as defined here; we thus decided to propose a simplified definition.

this implies that these can also be described in terms of equivalent GSs. In practice, however, simulating other automata via Turing machines will lead to rather complicated machine tables even for the simplest symbolic algorithms, and thus to unnecessarily complicated shift spaces. In fact, different automata implement different atomic operations, so that a Turing machine can require multiple computation steps to simulate a single computation step of another automaton, even when the automaton is computationally less powerful. Therefore, we introduce a novel shift space to which we shall henceforth refer as versatile shift (VS), which will allow us to represent automata configuration dynamics on dotted sequences in a more straightforward and parsimonious fashion, simulating it in real-time. Our construction essentially relies on a redefinition of the concept of dotted sequence. Above, the dot was only used as a mnemonic symbol without any functional implication. Now, we introduce the dot as a meta-symbol which can be concatenated with two words $v_1, v_2 \in \mathbf{A}^*$ through $v = v_1.v_2$. Let $\hat{\mathbf{A}}^*$ denote the set of these dotted words. Moreover, let $\mathbb{Z}^- = \{i \mid i < 0,\ i \in \mathbb{Z}\}$ and $\mathbb{Z}^+ = \{i \mid i \geq 0,\ i \in \mathbb{Z}\}$ the sets of negative and non-negative indices. We can then reintroduce the notion of a dotted sequence as follows. Let $s \in \mathbf{A}^{\mathbb{Z}}$ be a bi-infinite sequence of symbols such that $s = w_\alpha v w_\beta$ with $v \in \hat{\mathbf{A}}^*$ as a dotted word $v = v_1.v_2$ and $w_\alpha v_1 \in \mathbf{A}^{\mathbb{Z}^-}$ and $v_2 w_\beta \in \mathbf{A}^{\mathbb{Z}^+}$. Through this definition, the indices of s are inherited from the dotted word v and are thus not explicitly prescribed. Whereas GSs can only rewrite each symbol in their DoD with a new one, VSs are endowed with a more general rewriting operation, substituting dotted words in their DoD with other dotted words of equal or different lengths (as already hinted, yet not implemented, by Moore, 1990). This adds expressiveness to VSs, allowing for the parsimonious real-time simulation of a range of automata (see Figure 2 for a pictorial representation of the difference in substitution operations between GSs and VSs).

More formally, we define a VS as a pair $M_{VS} = (\mathbf{A}^{\mathbb{Z}}, \Omega)$, with $\mathbf{A}^{\mathbb{Z}}$ being the space of dotted sequences, and $\Omega : \mathbf{A}^{\mathbb{Z}} \to \mathbf{A}^{\mathbb{Z}}$ defined by

$$\Omega(s) = \sigma^{F(s)}(s \oplus G(s)) \tag{2}$$

with

$$\begin{aligned} &F : \mathbf{A}^{\mathbb{Z}} \to \mathbb{Z} \\ &\oplus : \mathbf{A}^{\mathbb{Z}} \times \mathbf{A}^{\mathbb{Z}} \to \mathbf{A}^{\mathbb{Z}} \\ &G : \mathbf{A}^{\mathbb{Z}} \to \mathbf{A}^{\mathbb{Z}}, \end{aligned} \tag{3}$$

where the operator "$\oplus$" substitutes the dotted word $v_1.v_2 \in \hat{\mathbf{A}}^*$ in s with

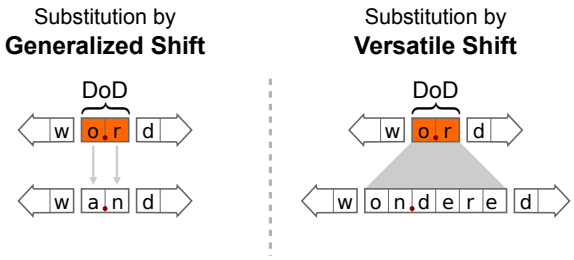


Figure 2: **Difference between substitution operation in generalized and versatile shifts.** In this Figure, we show two example substitutions by respectively a generalized shift and a versatile shift. While a generalized shift can only rewrite each symbol of the dotted word in its Domain of Dependence (DoD) with a new one, a versatile shift can substitute the dotted word in its DoD with any other arbitrary dotted word.

a new dotted word $\hat{v_1}.\hat{v_2} \in \hat{\mathbf{A}}^*$ specified by G, while $F(s) = F|_{\hat{\mathbf{A}}^*}(v_1.v_2)$ determines the number of shift steps as for the GS above. The action of F, G and $\oplus$ in the VS depends on a finite dotted sub-sequence $v_1.v_2$ inside the original dotted sequence $s = w_\alpha v w_\beta$, as determined by the DoD of the VS, again defined as a set of consecutive integers denoting cell positions on the original dotted sequence. The DoD of a GS can be specified by an open interval (k_l, k_r) on the integers, with $k_l \leq 0$ and $k_r \geq 0$. Additionally, for a DoD $= (k_l, k_r)$, it is useful to define $\text{DoD}_\alpha = (k_l, 0)$ and $\text{DoD}_\beta = (-1, k_r)$ to denote the left and right part of the complete DoD on dotted sequences $\alpha.\beta$, with $\text{DoD} = \text{DoD}_\alpha \cup \text{DoD}_\beta$. The set V of dotted words that can appear in the DoD of a VS is a subset of $\hat{\mathbf{A}}^*$, and can be defined as $V = \{v \mid v = v_1.v_2 \in \hat{\mathbf{A}}^*,\ |v_1| = |\text{DoD}_\alpha|,\ |v_2| = |\text{DoD}_\beta|\}$.

To illustrate how VSs act on dotted sequences, consider for example the dotted sequence "`wo.rd`", and define a VS $\Omega_{\text{ex}}$with

$$\text{DoD} = (-2, 1) = \{-1, 0\},$$
$$G : \begin{cases} \texttt{o.r} \mapsto \texttt{a.n} \\ \texttt{a.n} \mapsto \texttt{on.dere}, \end{cases}$$
$$F : \begin{cases} \texttt{o.r} \mapsto 0 \\ \texttt{a.n} \mapsto 1, \end{cases}$$

then, applying Ω_{ex} to "wo.rd" once yields

$$\begin{aligned}\Omega_{\text{ex}}(\texttt{wo.rd}) &= \sigma^{F(\texttt{wo.rd})}\big(\texttt{wo.rd} \oplus G(\texttt{wo.rd})\big)\\ &= \sigma^{F(\texttt{wo.rd})}\big(\texttt{wo.rd} \oplus \texttt{a.n}\big)\\ &= \sigma^{F(\texttt{wo.rd})}\big(\texttt{wa.nd}\big)\\ &= \sigma^{0}\big(\texttt{wa.nd}\big)\\ &= \texttt{wa.nd}\end{aligned}$$

and applying it again to the resulting "wa.nd" dotted sequence yields

$$\begin{aligned}\Omega_{\text{ex}}(\texttt{wa.nd}) &= \sigma^{F(\texttt{wa.nd})}\big(\texttt{wa.nd} \oplus G(\texttt{wa.nd})\big)\\ &= \sigma^{F(\texttt{wa.nd})}\big(\texttt{wa.nd} \oplus \texttt{on.dere}\big)\\ &= \sigma^{F(\texttt{wa.nd})}\big(\texttt{won.dered}\big)\\ &= \sigma^{1}\big(\texttt{won.dered}\big)\\ &= \texttt{wo.ndered}\end{aligned}$$

where the DoD of the input string has been highlighted for clarity (again, contrast this with the pictorial representation given in Figure 2). Note that a VS reduces to a GS in the special case when G always substitutes a dotted sequence with one of the same (finite) length in both the left and the right sub-sequences, as in the previous example where $\texttt{wo.rd} \oplus G(\texttt{wo.rd}) = \texttt{wo.rd} \oplus \texttt{a.n} = \texttt{wa.nd}$.

A point worth noting is that endowing VS with the rewriting capability extends the GS in the direction of semi-Thue systems (also known as string rewriting systems), a universal model of computation introduced by Axel Thue in 1914 (see chapter 7 of Davis et al., 1994). These rewriting systems play an important role, for example, in algebraic specifications of abstract data structures, equational programming, program transformation and automated theorem proving, where the conditional and successive application of a finite set of rewrite rules transforms a given symbolic structure.

2.1.2. Simulation of Various Automata by Versatile Shifts

We will now discuss how a range of automata can be simulated in real-time by VSs by choosing appropriate dotted sequence representations of machine configurations, and by constructing F and G to reproduce the machine's operations and their conditional application.

Finite-state machines. The finite-state machine (FSM) model of computation has been introduced by McCulloch and Pitts in 1943 and is widely used to describe systems in many application fields, ranging from computer science to engineering and biology, to name a few. At every step of a computation a FSM is in one of a finite set of states, and it can change its state as a result of an incoming input signal. More formally, a FSM can be defined as a 5-tuple $M_{\text{FSM}} = (Q, \mathbf{T}, q_0, F, \delta)$, where Q is a finite set of control states, $\mathbf{T}$ is the input alphabet, $q_0 \in Q$ is the starting state, $F \subseteq Q$ is a set of accept states, and $\delta : Q \times \mathbf{T} \to Q$ is a transition function defined as follows:

$$\delta : (q_t, d_{0_t}) \mapsto q_{t+1}, \tag{4}$$

where $q_t, q_{t+1} \in Q$ are states, and $d_{0_t} \in \mathbf{T}$ is an input symbol. At each computation step, a FSM reads its current state q_t, consumes (i.e. reads and discards) its current input symbol d_t, and transitions to a new state $q_{t+1} = \delta(q_t, d_t)$ as prescribed by its transition function. It is possible to encode FSM configurations on dotted sequences as

$$q_t \,.\, d_{0_t}\ d_{1_t}\ \ldots\ d_{n_t} \tag{5}$$

where q_t, d_{0_t} and $d_{1_t} \ldots d_{n_t}$ are respectively the state, input symbol, and the rest of the unconsumed input of the FSM at time t. A VS simulating a FSM in real-time can be constructed by defining the Domain of Dependence to be $\text{DoD} = (-2, 1) = \{-1, 0\}$, F to always map to 0, and G so that for all $q_t \in Q, d_t \in \mathbf{T}$:

$$G : q_t.d_{0_t} \mapsto q_{t+1}.\epsilon \tag{6}$$

where $q_{t+1} = \delta(q_t, d_{0_t})$.

Push-down automata and Context-Free Grammars. A push-down automaton (PDA) is a computing machine that has sequential access to its input and can manipulate a stack memory by popping and pushing symbols on top of it. More formally, a PDA can be defined as a 6-tuple $M_{\text{PDA}} = (Q, \mathbf{N}, \mathbf{T}, q_0, F, \delta)$, where Q is a finite set of control states, $\mathbf{N}$ is the stack alphabet, $\mathbf{T}$ is the input alphabet, $q_0 \in Q$ is the starting state, $F \subseteq Q$ is a set of accept states, and δ is a transition function. If $F = \varnothing$, the PDA accepts its input when both the input tape and the stack are empty, and it is thus said to accept by *empty stack*.

A Deterministic PDA is a PDA in which any configuration of the machine defines at most one transition. As the mapping of non-deterministic

automata computation to Neural Networks is outside the scope of this work, in what follows we will only discuss Deterministic PDAs. Determinism will thus be implied from this point on. The transition function of a PDA is defined as follows:

$$\delta : Q \times \mathbf{T} \cup \{\epsilon\} \times \mathbf{N} \to Q \times (\mathbf{N} \cup \{\epsilon\}). \tag{7}$$

At each computation step, a PDA consumes an input symbol, pushes or pops a symbol on the top of its stack, and changes state as prescribed by its transition function applied to the current state q_t, currently read input symbol d_{0_t}, and the current top-of-stack symbol s_{0_t}. In particular, if $s_{0_t} \dots s_{m_t}$ is the current content of the stack, transitions of the form

$$\delta : (q_t, d_{0_t}, s_{0_t}) \mapsto (q_{t+1}, \epsilon)$$

apply a pop operation, such that the new stack content becomes equal to $s_{1_t} \dots s_{m_t}$. Push operations are instead applied by transitions of the form

$$\delta : (q_t, d_{0_t}, s_{0_t}) \mapsto (q_{t+1}, s_{0_{t+1}}),$$

so that the updated stack contains the symbols $s_{0_{t+1}} s_{0_t} \dots s_{m_t}$. Finally, for transitions of the form

$$\delta : (q_t, \epsilon, s_{0_t}) \mapsto (q_{t+1}, \chi),$$

the PDA does not consume any input symbol (i.e. it does not access its input at all), but either pops its top-of-stack, if $\chi = \epsilon$, or pushes symbol χ, if $\chi \in \mathbf{N}$.

PDA configurations can be encoded on dotted sequences as follows:

$$\underbrace{s_{m_t} \ \dots \ s_{0_t}}_{s_t} \ q_t \ . \ \underbrace{d_{0_t} \ \dots \ d_{n_t}}_{d_t} \tag{8}$$

where q_t, d_t and s_t are respectively the state, the unconsumed input and the content of the stack of the automaton in reversed order at time t.

A VS simulating a PDA in real-time can be constructed from the PDA's transition function by defining the Domain of Dependence to be $\text{DoD} = (-3, 1) = \{-2, -1, 0\}$, F to always map to 0, and G so that, given $\delta : (q_t, \kappa, s_{0_t}) \mapsto (q_{t+1}, \chi)$,

$$G : \begin{cases} s_{0_t} \ q_t \ . \ \kappa \ \mapsto \ \epsilon \ q_{t+1} \ . \ \epsilon & \text{if } \chi = \epsilon \\ s_{0_t} \ q_t \ . \ \kappa \ \mapsto \ s_{0_t} \ \chi \ q_{t+1} \ . \ \epsilon & \text{otherwise.} \end{cases} \tag{9}$$

PDA recognize the class of languages generated by context-free grammars (CFG). PDA and CGFs are thus equivalent in power. A CFG specifies a language, i.e. a set of strings on some alphabet, by defining how its words can be constructed, moving from a distinguished starting symbol and applying substitution rules until a string of unsubstitutable symbols (terminals) is reached.

A CFG can be formally defined as a 4-tuple $G_{CF} = (\mathbf{N}, \mathbf{T}, R, \mathtt{S})$, where $\mathbf{N}$ is a set of non-terminal symbols, $\mathbf{T}$ is a set of terminal symbols, $R \subset \mathbf{N} \times (\mathbf{N} \cup \mathbf{T})^*$ a set of substitution rules and $\mathtt{S}$ a distinguished start symbol. In particular, each rule in R can be written as $X \to w$, with $X \in \mathbf{N}$ and $w \in (\mathbf{N} \cup \mathbf{T})^*$.

For example, let us define a CFG G_{ex} with $\mathbf{N} = \{\mathtt{S}\}$, $\mathbf{T} = \{\mathtt{(}, \mathtt{[}, \mathtt{)}, \mathtt{]}\}$, and R containing the rules

$$\begin{aligned} \mathtt{S} &\to \mathtt{(S)} \\ \mathtt{S} &\to \mathtt{[S]} \\ \mathtt{S} &\to \epsilon. \end{aligned}$$

Then G_{ex} generates the language $\mathcal{L}_{\text{ex}}$ of balanced round and square brackets. By applying the substitution rules we can in fact derive any string in that language. For illustration purposes, an example derivation would be: $\mathtt{S} \to \mathtt{[S]} \to \mathtt{[(S)]} \to \mathtt{[()]} \in \mathcal{L}_{\text{ex}}$. It is always possible to construct, given any CFG, a PDA recognizing its language, and viceversa.

Top-down recognizers. In one of the examples presented later in the text, we will make use of top-down recognizers (TDRs, see Aho and Ullman, 1972) that can process locally unambiguous non-left-recursive CFGs[3]. TDRs are a subclass of PDA that can simulate rule expansion to accept languages generated by non-left-recursive CFGs. Given any CFG G_{CF} that is not left-recursive, it is possible to construct a TDR that can parse strings belonging to the context-free language generated by that grammar. If the input string of a TDR constructed from G_{CF} is in the language generated by that grammar (and thus it can be derived by the grammar), then the TDR will end its

[3]A recursive CFG is a CFG including rules $A \to uAv$ that expand a non-terminal symbol A into a string containing the same non-terminal. A CFG is called left-recursive if such rules appear in the form $A \to Aw$. A CFG is locally unambiguous if there are no two rules expanding the same nonterminal.

computation with an empty stack and input, and is said to accept the string by empty stack. We are specifically interested in TDRs that process locally unambiguous CFGs, which have the additional property of needing only one state to perform their computation. To construct such a TDR from a locally unambiguous non-left-recursive CFG $G_{CF} = (\mathbf{N}, \mathbf{T}, R, \mathtt{S})$ it is sufficient to define its δ function in the following way:

$$\delta : \begin{cases} (q_0,\, a,\, a) & \mapsto & (q_0,\, \epsilon) & \text{for all } a \in \mathbf{T} \\ (q_0,\, \epsilon,\, X) & \mapsto & (q_0,\, w) & \text{for all } (X \to w) \in R \end{cases} \tag{10}$$

where $X \in \mathbf{N}$ is a non-terminal, $w \in (\mathbf{N} \cup \mathbf{T})^*$ is a string of terminals and non-terminals, and q_0 is the TDR's only state. Note that in the definition above we endow TDRs with the additional capability of pushing strings w on the stack rather than single symbols.

As our TDRs only have one state q_0, we can describe their machine configuration without referring to the current state. It is thus possible to encode TDR configurations on dotted sequences as follows:

$$\underbrace{s_{m_t} \; \ldots \; s_{0_t}}_{s_t} \; . \; \underbrace{d_{0_t} \; \ldots \; d_{n_t}}_{d_t} \tag{11}$$

where d_t and s_t are respectively the unconsumed input and the content of the stack of the automaton in reverse order at time t. Similarly, simpler VSs than those needed to simulate PDAs can be constructed from a TDR's transition function, by defining the Domain of Dependence to be DoD $= (-2, 1) = \{-1, 0\}$, F to always map to 0 and G to mirror Equation 10 so that

$$G : \begin{cases} a\,.a & \mapsto & \epsilon\,.\epsilon \\ X.a & \mapsto & w.\epsilon \end{cases} \tag{12}$$

for all $a \in \mathbf{T}$, $(X \to w) \in \mathbf{R}$.

Turing machines. A Turing machine (TM) is an automaton with read-write random access to a two-sided infinite tape (Turing, 1937; Sipser, 2006). TMs are central to the Theory of Computation, and they are thought to be powerful enough to model any physically realizable computation (with assumptions of unbounded resources). A TM has an in-built tape (doubly-infinite one dimensional memory with one symbol capacity at each memory location) and a finite-state controller endowed with a read-write head that follows the

instructions encoded by the transition function. At each step of the computation, given the current state and the current symbol read by the read-write head, the controller determines via a δ transition function the writing of a symbol on the current memory location, a shift of the read-write head to the memory location to the left ($\mathcal{L}$) or to the right ($\mathcal{R}$) of the current one, and the transition to a new state for the next computation step. Formally, a TM (Turing, 1937) can be defined as a 7-tuple $M_{\text{TM}} = (Q, \mathbf{N}, \mathbf{T}, q_0, \sqcup, F, \delta)$, where Q is a finite set of control states, $\mathbf{N}$ is a finite set of tape symbols also containing the blank symbol $\sqcup$, $\mathbf{T} \subset \mathbf{N} \setminus \{\sqcup\}$ is the input alphabet, $q_0 \in Q$ is the starting state, $F \subset Q$ is a set of 'halting' states reached at the end of the computation and $\delta : Q \times \mathbf{T} \to Q \times \mathbf{T} \times \{\mathcal{L}, \mathcal{R}\}$ is a partial transition function, the so-called machine table, that determines the dynamics of the machine. In particular, δ is defined as follows:

$$\delta : (q_t, d_{0_t}) \mapsto (q_{t+1}, d_{0_{t+1}}, m) \tag{13}$$

where $q_t, q_{t+1} \in Q$ are the state of the machine before and after the transition, $d_{0_t}, d_{0_{t+1}} \in \mathbf{N}$ are respectively the read and rewritten symbol, and $m \in \{\mathcal{L}, \mathcal{R}\}$ denotes the shift of the read-write head to the left or to the right.

At a given computation step, the content of the tape together with the position of the read-write head and the current controller state define a machine configuration. It is possible to encode TM configurations on dotted sequences as follows:

$$s = \underbrace{\ldots\ d_{-2_t}\ d_{-1_t}}_{l_t}\ q_t\ .\ \underbrace{d_{0_t}\ d_{1_t}\ d_{2_t}\ \ldots}_{r_t}, \tag{14}$$

where l_t describes the part of the tape to the left of the read-write head, r_t describes the part to its right, q_t describes the current state of the machine controller, and the central dot denotes the current position of the read-write head, i.e. d_{0_t}, the symbol to its right.

A VS simulating a TM in real-time can be constructed from the TM's transition function by defining the Domain of Dependence to be $\text{DoD} = (-3, 1) = \{-2, -1, 0\}$, and G and F so that, given $\delta : (q_t, d_{0_t}) \mapsto (q_{t+1}, \hat{d}_{0_t}, m)$,

$$\begin{aligned}
G: &\ \begin{cases} d_{-1_t}\ q_t\ .\ d_{0_t} \ \mapsto\ d_{-1_t}\ \hat{d}_{0_t}\ .\ q_{t+1} & \text{if } m = \mathcal{R} \\ d_{-1_t}\ q_t\ .\ d_{0_t} \ \mapsto\ q_{t+1}\ d_{-1_t}\ .\ \hat{d}_{0_t} & \text{if } m = \mathcal{L} \end{cases} \\
F: &\ \begin{cases} d_{-1_t}\ q_t\ .\ d_{0_t} \ \mapsto\ -1 & \text{if } m = \mathcal{R} \\ d_{-1_t}\ q_t\ .\ d_{0_t} \ \mapsto\ +1 & \text{if } m = \mathcal{L} \end{cases}
\end{aligned} \tag{15}$$

for all $d_{-1_t} \in \mathbf{N}$.

The following example will clarify how the VS defined as above (Equation 15) can simulate a TM. Consider, for instance, the dotted sequence "$\mathtt{w}q_0\mathtt{.ord}$", and define a TM such that $\delta : (q_0, \mathtt{o}) \mapsto (q_1, \mathtt{a}, \mathcal{R})$ and $\delta : (q_1, \mathtt{r}) \mapsto (q_1, \mathtt{n}, \mathcal{L})$. Then a computation step of the TM starting from the "$\mathtt{w}q_0\mathtt{.ord}$" configuration would yield a new configuration "$\mathtt{wa}q_1\mathtt{.rd}$"; by running the TM again, this time starting from "$\mathtt{wa}q_1\mathtt{.rd}$", a computation step would yield "$\mathtt{w}q_1\mathtt{.and}$", as prescribed by the transition function we defined. Constructing a VS Ω_{ex} as specified by Equation 15 and applying it to "$\mathtt{w}q_0\mathtt{.ord}$":

$$\begin{aligned}
\Omega_{\text{ex}}(\mathtt{w}q_0\mathtt{.ord}) &= \sigma^{F(\mathtt{w}q_0\mathtt{.ord})}\big(\mathtt{w}q_0\mathtt{.ord} \oplus G(\mathtt{w}q_0\mathtt{.ord})\big) \\
&= \sigma^{-1}\big(\mathtt{w}q_0\mathtt{.ord} \oplus \mathtt{wa}.q_1\big) \\
&= \sigma^{-1}\big(\mathtt{wa}.q_1\mathtt{rd}\big) \\
&= \mathtt{wa}q_1\mathtt{.rd}
\end{aligned} \tag{16}$$

and by applying it again to the resulting "$\mathtt{wa}q_1\mathtt{.rd}$" dotted sequence we obtain

$$\begin{aligned}
\Omega_{\text{ex}}(\mathtt{wa}q_1\mathtt{.rd}) &= \sigma^{F(\mathtt{wa}q_1\mathtt{.rd})}\big(\mathtt{wa}q_1\mathtt{.rd} \oplus G\big(\mathtt{wa}q_1\mathtt{.rd}\big) \\
&= \sigma^{+1}\big(\mathtt{wa}q_1\mathtt{.rd} \oplus q_0\mathtt{a.n}\big) \\
&= \sigma^{+1}\big(\mathtt{w}q_0\mathtt{a.nd}\big) \\
&= \mathtt{w}q_0\mathtt{.and}
\end{aligned} \tag{17}$$

where the DoD of the input string to the VS has been highlighted for clarity. Note that the dotted representation of the machine configuration requires index -1 to always contain the machine state. For this reason, it is not enough to only rewrite the symbols in $\{-1, 0\}$ (i.e. the machine state and the current symbol under the read-write head) to simulate a TM, as intuition would instead suggest. In fact, a VS first applies a rewriting of its DoD, and then shifts the resulting dotted sequence to the left (when $F(s) = -1$) or the right (when $F(s) = +1$). In particular, the shift is needed to simulate the movement of the read-write head on the machine tape. In order to make sure that at the end of the substitution and shift the machine state is correctly placed at its reserved index -1, the substitution must leave it displaced one place to the right if a left shift is to be applied (as in Equation 16), or one to the left in case of a right shift (as in Equation 17). This last case requires the additional dependence of the VS on index -2. Furthermore, note that our construction is equivalent to that from Moore (1990, 1991): the VS defined

in Equation 15 is nothing more than the GS introduced by Moore to prove the equivalence between GSs and TMs.

2.2. Introducing Nonlinear Dynamical Automata

We will now discuss how VSs, and thus the models of symbolic computation they can simulate, can be mapped to piecewise affine-linear systems on a vectorial space, obtaining nonlinear dynamical automata.

2.2.1. Gödel Encodings and the Symbol Plane

A Gödel encoding (or Gödelization, see Gödel, 1931) allows one to uniquely assign a real number to a sequence such that the space of one-sided infinite sequences can be mapped to the real interval $[0, 1]$.[4] For completeness, Gödelization is subsequently discussed alongside its graphical representation, provided in Figure 3.

Let $\mathbf{A}^{\mathbb{N}}$ be the space of one-sided infinite sequences over an alphabet $\mathbf{A}$ containing $|\mathbf{A}| = g$ symbols, and $s = d_1 d_2 \ldots$ a sequence in this space, with d_k being the k-th symbol in s. Additionally, let $\gamma : \mathbf{A} \to \mathbb{N}$ be a one-to-one function associating each symbol in the alphabet $\mathbf{A}$ with a natural number. Then a Gödelization is a mapping from $\mathbf{A}^{\mathbb{N}}$ to $[0, 1] \subset \mathbb{R}$ defined as follows:

$$\psi(s) := \sum_{k=1}^{\infty} \gamma(d_k) g^{-k}. \tag{18}$$

Conveniently, Gödelization can also be employed on a dotted sequence $\alpha.\beta \in \mathbf{A}^{\mathbb{Z}}$ — herein representing a machine configuration — by splitting it into its two one-sided constituents α' (the reversed α) and β. Defining two Gödel encodings ψ_x and ψ_y for α' and β respectively, induces a two-dimensional representation for $\alpha.\beta$, i.e. $\big(\psi_x(\alpha'), \psi_y(\beta)\big)$, known as symbol plane or symbologram, which is contained in the unit square $[0, 1]^2 \subset \mathbb{R}^2$. In encoding dotted sequences $\alpha.\beta$ representing configurations of the machines we consider in this paper, α often only ever contains states as first symbols, and

[4] A Gödel encoding maps sequences on some alphabet $\mathbf{A}$ to real numbers through the use of a base-b expansion, with $b = |\mathbf{A}|$. It can be proven that any base-b expansion represents a real number, and that any real number has a unique base-b representation under a weak condition. The uniqueness of the Gödel encoding (and decoding) of any sequence follows from the same proof.

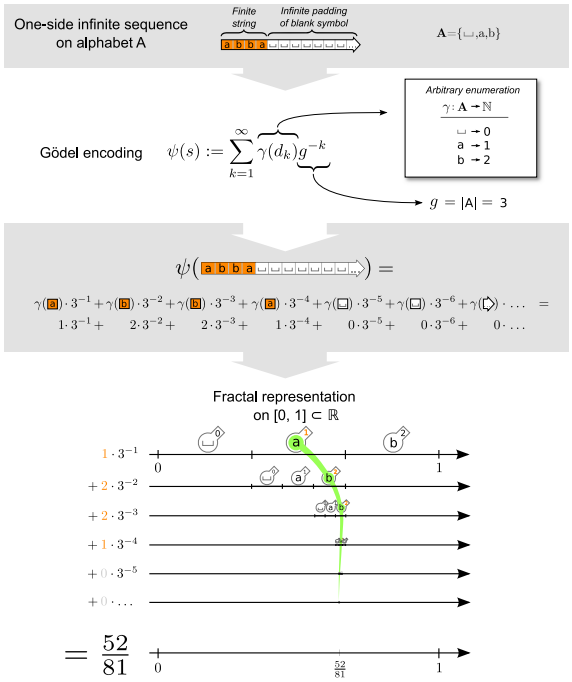


Figure 3: **Three representations of the Gödel Encoding of a sequence.** The first one is just the definition of the Gödel encoding, with details on the specific choice of the enumerating γ function and the induction of the g constant, given the alphabet $\mathbf{A}$ from which the sequence takes its symbols. The second one is an expansion of the series in the definition. The third one visually conveys the fractal and convergent nature of the series, highlighting the relation between numbers and symbols by the use of the color orange. At each level of this representation, from top to bottom, the encoding of the sequence "`abba`␣␣␣..." is sequentially constructed, highlighting the contribution of each encoded symbol to the real number resulting from the complete Gödelization.

tape symbols in the rest of the sequence. In this case we can define a more refined Gödelization that covers all of the representational space $[0, 1] \in \mathbb{R}$:

$$\psi_x(s) := \gamma_q(d_1)n_q^{-1} + \sum_{k=1}^{\infty} \gamma_s(d_{k+1})n_s^{-k}n_q^{-1}, \tag{19}$$

where γ_q and γ_s respectively enumerate the set of states Q and the tape alphabet $\mathbf{A}$, and where $n_q = |Q|$, $n_s = |\mathbf{A}|$.

2.2.2. Versatile Shifts as Affine-Linear Transformations

Push and pop operators can be defined on one-sided infinite sequences $\mathbf{A}^{\mathbb{N}}$ on some alphabet $\mathbf{A}$. The push operator $\odot$ is defined so that $s \odot b$ adds the contents of a word $b \in \mathbf{A}^*$ to the beginning of $s \in \mathbf{A}^{\mathbb{N}}$, whereas the pop operator $\ominus$ is defined so that $\ominus^p s$ removes the first p symbols in s. We will now show that Gödelizing a sequence resulting from the application of pop and push operations is equivalent to applying an affine-linear transformation on the original Gödelized sequence. We will then show that VSs on a dotted sequence $\alpha.\beta$ can be mapped to push and pop operations on its one-sided constituents α' and β. Let $s = d_1d_2d_3\ldots$ be a one-sided infinite sequence on an alphabet A. Applying a pop operation $\ominus^p$ to s yields $\ominus^p s = d_{p+1}d_{p+2}d_{p+3}\ldots$, while pushing a word $b = b_1 \ldots b_r$ to the beginning of s yields $s \odot b = b_1 \ldots b_r d_1 d_2 \ldots$. In this case

$$\psi\left(s\right) = \gamma(d_1)g^{-1} + \gamma(d_2)g^{-2} + \gamma(d_3)g^{-3} + \ldots,$$

so that

$$\begin{aligned}\psi(\ominus^p s) &= \gamma(d_{p+1})g^{-1} + \gamma(d_{p+2})g^{-2} + \gamma(d_{p+3})g^{-3} + \ldots \\ &= \psi(s) \cdot g^p - \sum_{i=1}^{p} \gamma(d_i)g^{p-i},\end{aligned}$$

and

$$\begin{aligned}\psi(s \odot b) &= \gamma(b_1)g^{-1} + \ldots + \gamma(b_r)g^{-r} + \\ &\quad \gamma(d_1)g^{-(r+1)} + \gamma(d_2)g^{-(r+2)} + \ldots \\ &= \psi(s) \cdot g^{-r} + \sum_{i=1}^{r} \gamma(b_i)g^{-i},\end{aligned}$$

proving that the resulting Gödelized sequences can be obtained by applying affine-linear transformations to the original Gödelized sequences. For both pop and push operations, the parameters of the affine-linear transformations only depend on the number and identities on the symbols that are respectively removed from or added to the beginning of the original sequence. This is of particular importance in the framework of interactive computation (Wegner, 1998), where the newly added symbol stems from the network's interaction with its environment. Accordingly, the symbol b becomes represented by a linear operator acting on the system's state space, analogous to quantum operators acting on Hilbert spaces (beim Graben et al., 2008).

As previously discussed, a VS defines two operations on dotted sequences, a substitution operation $s \oplus G(s)$ which replaces the dotted sub-sequence in the DoD of the shift with a new dotted sequence $G(s)$, and a shift operation $\sigma^{F(s)}$ shifting the symbols in s to the left or to the right by $F(s)$ positions. Let $s \oplus G(s) = w_\alpha u.v w_\beta \oplus \hat{u}.\hat{v}$ be a substitution replacing the dotted sub-sequence $u.v$ in s with the dotted word $\hat{u}.\hat{v}$, then $s \oplus G(s)$ can be straightforwardly mapped to pop and push operations on $u' {w_\alpha}'$ and $v w_\beta$, the one-sided constituents of the original dotted sequence s, as follows:

$$\begin{aligned} w_\alpha u.v w_\beta \oplus \hat{u}.\hat{v} &= \left((\ominus^{|u'|} u' {w_\alpha}') \odot \hat{u}'\right)'.\left((\ominus^{|v|} v w_\beta) \odot \hat{v}\right) \\ &= \left({w_\alpha}' \odot \hat{u}'\right)'.\left(w_\beta \odot \hat{v}\right) \\ &= w_\alpha \hat{u}.\hat{v} w_\beta \end{aligned}$$

showing that substitutions on dotted sequences can be mapped to pop and push operations on its one-sided constituents. A left shift σ^{-1} and a right shift σ^1 on a dotted sequence $\alpha.\beta = \ldots\ d_{-2}\ d_{-1}\ .\ d_0\ d_1\ \ldots$ can be mapped to push and pop operations on its one-sided constituents as follows:

$$\begin{aligned} \sigma^{-1}(\ldots\ d_{-2}\ d_{-1}\ .\ d_0\ d_1\ \ldots) &= (\alpha' \odot d_0)'.(\ominus^1 \beta) \\ &= \ldots\ d_{-1}\ d_0\ .\ d_1\ d_2\ \ldots\ , \end{aligned}$$

and

$$\begin{aligned} \sigma^{1}(\ldots\ d_{-2}\ d_{-1}\ .\ d_0\ d_1\ \ldots) &= (\ominus^1 \alpha')'.(\beta \odot d_{-1}) \\ &= \ldots d_{-3}\ d_{-2}\ .\ d_{-1}\ d_0\ \ldots\ , \end{aligned}$$

showing that shifts on dotted sequences can be mapped to pop and push operations on its one-sided constituents. Any arbitrary shift σ^k with $k \in \mathbb{Z}$ can be

obtained by composition of left and right shifts; as the composition of affine-linear transformations is an affine-linear transformation, the Gödelization of a sequence resulting from the composition of shift operations is equivalent to an affine-linear transformation on the original Gödelized sequence. We have thus shown that VSs on dotted sequences can be mapped to pop and push operations on one-sided infinite sequences, and that the Gödelization of these operations can be mapped to affine-linear transformations on the original sequences. On the symbologram, each substitution and shift operation on a Gödelized dotted sequence $\alpha.\beta$ by a VS involves two affine-linear transformations, one acting on the Gödelized α' (the reversed α) and one on the Gödelized β. The parameters of the affine-linear transformations only depend on the symbols of the dotted sequence in the DoD of the VS. All dotted sequences which share the same DoD symbols are thus associated to the same pair of affine-linear transformations. For this reason, the symbologram representation of VSs leads to piecewise affine-linear maps on rectangular partitions of the unit square, referred to as a nonlinear dynamical automata (Tabor, 2000; Tabor et al., 2013; beim Graben et al., 2004, 2008).

2.2.3. Nonlinear Dynamical Automata

A nonlinear dynamical automaton (NDA) is a triple $M_{NDA} = (X, P, \Phi)$, where P is a rectangular partition of the unit square $X = [0,1]^2 \subset \mathbb{R}^2$, that is

$$P = \{D^{i,j} \subset X |\ 1 \leq i \leq m,\ 1 \leq j \leq n,\ m, n \in \mathbb{N}\}, \tag{20}$$

so that each cell is defined as $D^{i,j} = I_i \times J_j$, with $I_i, J_j \subset [0,1]$ being real intervals for each bi-index (i,j), with $D^{i,j} \cap D^{k,l} = \varnothing$ if $(i,j) \neq (k,l)$, and $\bigcup_{i,j} D^{i,j} = X$. The couple (X, Φ) is a time-discrete dynamical system with phase space X and the flow $\Phi : X \to X$ is a piecewise affine-linear map such that $\Phi_{|D^{i,j}} := \Phi^{i,j}$, with $\Phi^{i,j}$ having the following form:

$$\Phi^{i,j}(\mathbf{x}) = \begin{pmatrix} a_x^{i,j} \\ a_y^{i,j} \end{pmatrix} + \begin{pmatrix} \lambda_x^{i,j} & 0 \\ 0 & \lambda_y^{i,j} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}. \tag{21}$$

Note that the NDA, as any piecewise affine-linear system, also requires a switching rule $\Theta(x,y) \in \{(i,j) |\ 1 \leq i \leq m, 1 \leq j \leq n\}$, which selects the appropriate branch, and thus dynamics (i.e. $\Phi(x,y) = \Phi^{i,j}(x,y) \iff \Theta(x,y) = (i,j)$). A mapping between a VS and a NDA can be defined following the methods outlined in Section 2.2.1 and Section 2.2.2, therefore enabling the derivation of the parameters of the NDA. That is, first each cell

$D^{i,j} = I_i \times J_j$ can be seen as containing all the Gödelized dotted sequences $\alpha.\beta$ which agree (i.e. have the same symbols) in the Domain of Dependence. In particular, the I_i interval contains all the DoD-agreeing Gödelized α' (the reversed α) sub-sequences, whereas the J_j interval contains all the DoD-agreeing Gödelized β sub-sequences. This leads to a partition of the unit square with a number i of I intervals equal to the number of possible one-sided sub-sequences that can appear in the left DoD of the VS, and a number j of J intervals equal to the number of possible one-sided sub-sequences that can appear in the right DoD. For example, for a VS simulating a FSM, the left Domain of Dependence $\text{DoD}_\alpha = \{-1\}$ of the dotted sequences representing machine configurations only ever contains states, and the right Domain of Dependence $\text{DoD}_\beta = \{0\}$ only ever contains input symbols. In this case the number of I_i intervals becomes equal to the number of states $n_q = |Q|$ in the FSM, and the number of J_j intervals equal to the number of input symbols $n_s = |\mathbf{T}|$, where Q and $\mathbf{T}$ are respectively the set of states and that of input symbols in the FSM. For a VS simulating a TM, instead, the left Domain of Dependence $\text{DoD}_\alpha = \{-2, -1\}$ only ever contains states at index -1, and tape symbols at index -2, and the right Domain of Dependence $\text{DoD}_\beta = \{0\}$ always contains tape symbols. This leads to a partition of the unit square with a number of I_i intervals equal to $m = n_q n_s$, and one of J_j intervals equal to $n = n_s$, leading to a total of $n_q n_s^2$ cells, where n_s is the number of symbols in the tape alphabet $\mathbf{N}$ and n_q is the number of states in Q.

Following Section 2.2.2, substitutions and shifts on a sequence can be mapped to affine-linear transformations on its Gödelization. For this reason, each cell in the partition P of the unit square is associated with a different affine-linear transformation with parameters $(a_x^{i,j}, a_y^{i,j})$ and $(\lambda_x^{i,j}, \lambda_y^{i,j})$, which can be derived using the methods outlined in Section 2.2.2. Therefore a model of computation can be represented as a NDA by means of its Gödelized VS representation.

2.3. Solution Map between NDA and R-ANNs

The design of the map between the NDA and a first order R-ANN follows a conceptually natural and simple solution, which attempts to mimic the affine-linear dynamics (given by Equation 21) of the NDA on the partitioned unit square (see Carmantini et al., 2015 for preliminary work in this direction).

Let $\rho(\cdot)$ denote the proposed map. The objective is to map the orbits of the NDA (i.e. $\Phi^{i,j}(x, y)$) to orbits of the R-ANN, denoted as $\zeta^{i,j}(x, y)$. The role of ρ is to encode both the affine-linear dynamics within each partition cell

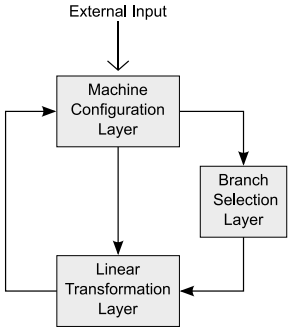


Figure 4: **Connectivity between neural layers within the network.** The machine configuration layer (MCL) receives external input (in this case the encoded initial machine configuration), and synaptically couples to the branch selection layer (BSL) and linear transformation layer (LTL). The BSL feed-forwards to the LTL and finally the LTL recurrently feedbacks to the MCL, where the output is read-out.

$(D^{i,j})$ and to emulate the transitions from cell to cell by suitably activating certain neural units within the R-ANN. To achieve this, we propose a network architecture with three layers, namely a machine configuration layer (MCL), a branch selection layer (BSL) and a linear transformation layer (LTL), as depicted in Figure 4. Therefore, we generically define the proposed map as follows:

$$\zeta = \rho(\mathcal{I}, \mathcal{A}, \Phi, \Theta), \tag{22}$$

where $\mathcal{I}_{2\times 2}$ is the identity matrix that maps (identically) the initial conditions of the NDA to the R-ANN and $\mathcal{A}$ is the synaptic weight matrix that defines the network architecture, which will be discussed in subsequent Sections. In addition, ρ generates different neural dynamics for each type of the neural units, i.e. $\zeta = (\zeta_1, \zeta_2, \zeta_3)$, corresponding to MCL, BSL and LTL, respectively. The details of the R-ANN architecture and its dynamics will now be presented.

2.3.1. Network Architecture and Neural Dynamics

The simulation of a NDA orbit within the R-ANNs is distributed among MCL, BSL and LTL. Since $\Phi^{i,j}(x)$ is a two-dimensional de-coupled discrete map it suggests only two neural units in a read-out layer, which is a role taken by the MCL. We refer to the two MCL units as c_x and c_y. At each

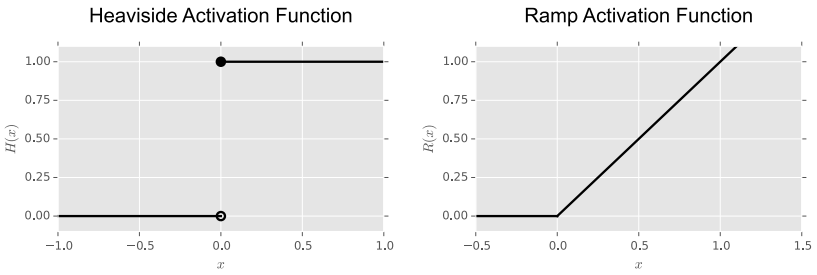


Figure 5: **Activation functions employed in the network.** In particular, the Heaviside function $H(x)$ is employed by units in the branch selection layer and the Ramp function $R(x)$ is used in both machine configuration layer and linear transformation layer.

computation step the MCL stores the encoding of the current machine configuration, which is then passed on to the BSL and LTL units. Subsequently, two sets of BSL units (b_x and b_y) functionally act as a switching system that determines to which cell $D^{i,j}$ the current machine configuration belongs, triggering the appropriate units within two sets of LTL units (t_x and t_y), effectively emulating the application of an affine-linear transformation $\Phi^{i,j}$ on an encoded machine configuration. This action corresponds to the application of a symbolic operation by the original machine, leading to a configuration update. The result of the transformation is then fed back to the MCL, representing the configuration (i.e. the machine's symbolic data) for the next computation step. These successive transformations effectively emulate the action of a NDA, where for every computational step an affine-linear transformation is applied to the values encoding the representation of the machine configuration. The neural units in the various layers make use of either the Heaviside (H) or the Ramp (R) activation functions defined as follows (see also Figure 5):

$$H(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \qquad R(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases} \tag{23}$$

2.3.2. Machine Configuration Layer

The MCL encodes the state of the simulated NDA, and thus the data of the simulated automaton, while acting as a read-out neural layer. At

the same time it mediates at each computation step the transmission of the current Gödel encoding of the emulated machine's configuration to the BSL and LTL units. Since the Gödel encoding of a dotted sequence representing a machine configuration consists of two values (see Section 2.2.1), this implies that the MCL solely requires two neural units (c_x and c_y) to code for the current configuration. As a consequence, the initialization of the R-ANNs is performed in this layer, where the initial conditions $(\psi_x(\alpha'), \psi_x(\beta))$ are identically transformed (via $\mathcal{I}$) by the map $\rho(\cdot)$ as follows:

$$(c_x, c_y) = (\psi_x(\alpha'), \psi_x(\beta)) \equiv \zeta_1 = \rho(\mathcal{I}, \cdot, \cdot, \cdot)_{|(\psi_x(\alpha'), \psi_x(\beta))} \tag{24}$$

Following every computation step, these neural units receive inputs from the LTL units and are subsequently activated via the ramp activation function (Equation 23); in other words $\zeta_1 \equiv (c_x, c_y) = (R(\sum_i t_x^i), R(\sum_j t_y^j))$. Finally, these synaptically project onto the BSL and LTL neural units (refer to Figure 6 for details of the connectivity).

2.3.3. Branch Selection Layer

The BSL acts as a control unit that enables the sequential mapping of the orbits of the NDA, $\Phi^{i,j}(x, y)$, to orbits of the R-ANNs, $\zeta^{i,j}(x, y)$. Specifically, the BSL functionally embodies the switching rule $\Theta(x, y)$ and coordinates the dynamic switching between LTL units. Sequentially, under the action of BSL units, only a single pair of LTL units $(t_x^{i,j}, t_y^{i,j})$ dedicated to emulate $\Phi^{i,j}$ become active, which then operate on an encoded Machine configuration. In particular, the BSL units make sure that $(t_x^{i,j}, t_y^{i,j})$ become active only if $(c_x, c_y) \in D^{i,j} = I_i \times J_j$, with $I_i = [\xi_i, \xi_{i+1})$ being the i-th interval on the x-axis and $J_j = [\eta_j, \eta_{j+1})$ being the j-th interval on the y-axis. The switching rule is mapped by $\rho(\cdot)$ as follows:

$$\zeta_2(x, y) = \rho(\cdot, \cdot, \cdot, \Theta(x, y) = \{i, j\}) \tag{25}$$

The implementation of $\zeta_2(x, y)$ is mediated by two sets of neural units, i) the b_x set with m units (the number of I intervals on the x-axis) and ii) the b_y set with n units (the number of J intervals on the y axis), which are activated via a Heaviside activation function (Equation 23) after receiving excitatory inputs with synaptic weight 1 from the MCL layer (i.e. c_x and c_y units) in the following way:

$$\begin{aligned} b_x^i &= H(c_x - \xi^i) \qquad \text{with} \qquad \xi^i = \min(I_i), \\ b_y^j &= H(c_y - \eta^j) \qquad \text{with} \qquad \eta^j = \min(J_j). \end{aligned} \tag{26}$$

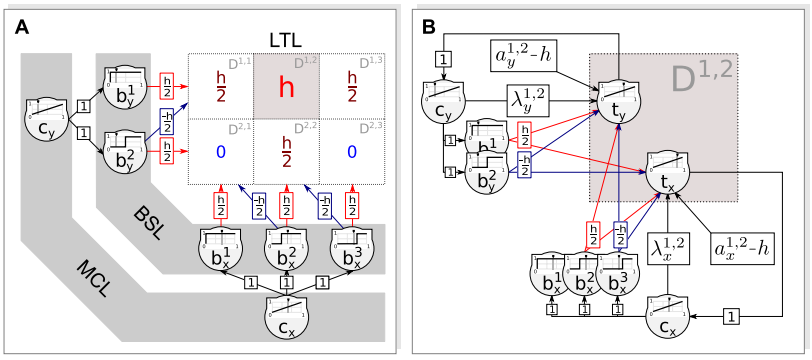


Figure 6: **Detailed feedforward connectivity and weights for neural network simulating a nonlinear dynamical automaton with only 6 branches.** (A) The machine configuration layer (MCL) units (c_x, c_y) feed-forward connect to all the branch selection layer (BSL) units with weight of 1. Every BSL unit excites with weights $\frac{h}{2}$ (in red) and also inhibits with weights $\frac{h}{2}$ (in blue) the relevant linear transformation layer (LTL) units contained within each cell (as indicated by the red and blue arrows respectively). Each cell D^{ij} indicates the overall summed input value received by each LTL unit (for visualization purpose/convenience not shown) from the BSL. In this case only the LTL units in cell $D^{1,2}$ are activated with overall BSL input value of h (red). (B) A zoom-in of panel (A), shows in detail how each pair of LTL units contained within each cell (in this case $D^{1,2}$) receives inputs from the MCL and BSL units as shown. In addition, the LTL units may have internal dynamics described by parameter a (equivalently, this can be seen as input from an always-active unit). To actually produce output, the overall input to an LTL unit must overcome its internal h inhibition. Upon activation, the LTL unit's output is fed back to the paired MCL unit with weight of 1.

That is, the activation of the BSL units depends on a threshold, implemented here as a synaptic projection from an always-active bias unit, that is defined as the minimum of the intervals I_i and J_j respectively for the b_x^i and b_y^i units. This has the effect of centering the threshold towards the left boundary of each interval (i.e. a bias of $-\xi^i$ for b_x^i unit and $-\eta^j$ for b_y^j). Therefore, if the read-out (i.e. encoded machine configuration) of the c_x and c_y units in the MCL corresponded to a point on the unit square belonging to cell $D^{i,j}$, then the b_x^i unit would be triggered active as well as all units b_x^k with $k < i$. The same would occur for neurons b_y^j and all neural units b_y^k with $k < j$.[5] Upon excitation, these BSL units then synaptically project to the relevant LTL units, $(t_x^{i,j}, t_y^{i,j})$ that are naturally inactive due to a strong inhibitory bias with magnitude h (the role and value of h will be clarified in the subsequent Section). Specifically, each neural unit b_x^i establishes synaptic excitatory connections (with weight $\frac{h}{2}$) to all LTL units within the cells $D^{k,i}$ (i.e. $(t_x^{k,i}, t_y^{k,i})$) and also project with synaptic inhibitory connections (with weight $\frac{-h}{2}$) to all LTL units within the cells $D^{k,i-1}$ (i.e. $(t_x^{k,i-1}, t_y^{k,i-1})$), where $k = 1, \dots m$; for a graphical depiction see Figure 6. Similarly, each neural unit b_y^j projects with synaptic excitatory connections (with weight $\frac{h}{2}$) to all LTL units within the cells $D^{j,k}$ (i.e. $(t_x^{j,k}, t_y^{j,k})$) and also projects with synaptic inhibitory connections (with weight $\frac{-h}{2}$) to all LTL units within the cells $D^{j-1,k}$ (i.e. $(t_x^{j-1,k}, t_y^{j-1,k})$), where $k = 1, \dots n$; see Figure 6. The combined effect of the b_x^i units and b_y^j is therefore to counterbalance through their synaptic weights the natural inhibition (of bias h) of the LTL units in cell $D^{i,j}$. In other words each couple of LTL units $(t_x^{i,j}, t_y^{i,j})$ receives an input $B_x^i + B_y^j$, defined as follows:

$$\begin{aligned} B_x^i &= b_x^i \frac{h}{2} + b_x^{i+1} \frac{-h}{2} \\ B_y^j &= b_y^j \frac{h}{2} + b_y^{j+1} \frac{-h}{2}, \end{aligned} \tag{27}$$

where the input sum

$$B_x^i + B_y^j = \begin{cases} h & \text{if } (c_x, c_y) \in D_{i,j} \\ \frac{h}{2} & \text{if } c_x \in I_i, c_y \notin J_j \quad \text{or} \quad c_x \notin I_i, c_y \in J_j \\ 0 & \text{if } (c_x, c_y) \notin D_{i,j} \end{cases} \tag{28}$$

[5] Note that the action of the BSL could be equivalently implemented by interval indicator functions represented as linear combinations of Heaviside functions.

only triggers the relevant LTL unit if it reaches the value h. That is, if the pair $(t_x^{i,i}, t_y^{i,j})$, is selected by the BSL units (and thus $(c_x, c_y) \in D_{i,j}$), then $B_x^i + B_y^j = h$. Otherwise $B_x^i + B_y^j$ is either equal to $\frac{h}{2}$ or 0. An example of this mechanism is shown in Figure 6, where the LTL units in cell $D^{1,2}$ are activated via mediation of $b_x = \{b_x^1, b_x^2, b_x^3\}$ and $b_y = \{b_y^1, b_y^2\}$. Here, both b_x^3 and b_y^2 are not excited since respectively c_x and c_y are not activated enough to drive them towards their threshold. However, b_x^2 excites (with weights $\frac{h}{2}$) the LTL units in cell $D^{2,2}$ and $D^{1,2}$ and inhibits (with weights $\frac{-h}{2}$) the LTL units in cell $D^{2,1}$ and $D^{1,1}$. Equally, b_y^2 excites (with weights $\frac{h}{2}$) the LTL units in cell $D^{2,1}$, $D^{2,2}$ and $D^{2,3}$ and inhibits (with weights $\frac{-h}{2}$) the LTL units in cells $D^{1,1}$, $D^{1,2}$ and $D^{1,3}$. The b_x^1 and b_y^1 units excite respectively cells $\{D^{2,1}, D^{1,1}\}$ and $\{D^{1,1}, D^{1,2}, D^{1,3}\}$, but these do not inhibit any cells (due to boundary conditions).

2.3.4. Linear Transformation Layer

The LTL embodies the set of affine-linear transformations of the NDA from which the network is constructed, and thus the set of symbolic operations defined by the transition table of the simulated automaton. This endows the LTL with the functional ability of generating an updated encoded machine configuration from the current one. That is, the affine-linear transformation of a NDA, $\Phi^{i,j}(x, y) = (\lambda_x^{i,j} x + a_x^{i,j}, \lambda_y^{i,j} y + a_y^{i,j})$ within a cell $D^{i,j}$ is simulated by the LTL unit $(t_x^{i,j}, t_y^{i,j})$. This induces the following mapping:

$$(t_x^{i,j}, t_y^{i,j}) \quad = \quad \zeta_3^{i,j}(x, y) = \rho(\cdot, \cdot, \Phi^{i,j}(x, y), \cdot). \tag{29}$$

This affine-linear transformation is implemented in the form of synaptic computation, which is only triggered when the BSL units provide enough excitation enabling the two neural units $(t_x^{i,j}, t_y^{i,j})$ to cross their threshold value and execute the operation. The read-out of this process is as follows:

$$\begin{array}{rcl} t_x^{i,j} & = & R(\lambda_x^{i,j} c_x + a_x^{i,j} - h + B_x^i + B_y^j) \\ t_y^{i,j} & = & R(\lambda_y^{i,j} c_y + a_y^{i,j} - h + B_x^i + B_y^j), \end{array} \tag{30}$$

that is, initially the LTL units are rendered inactive with a strong inhibition bias h implemented as a synaptic projection from a bias unit, which is defined as follows:

$$-\frac{h}{2} \leq -\max_{i,j,k}(a_k^{i,j} + \lambda_k^{i,j}) \quad \text{with} \quad k = \{x, y\}. \tag{31}$$

This results from the fact that each BSL inputs B_x^i and B_y^i contribute respectively to half of the necessary excitation ($\frac{h}{2}$), that sum up and counterbalance the LTL's natural inhibition (refer to Equation 27 and Equation 28). The LTL units also receive inputs from the MCL units (c_x, c_y), which are respectively modulated by the synaptic weights $(\lambda_x^{i,j}, \lambda_y^{i,j})$ and once the LTL units cross their threshold (mediated by the ramp activation function) then the intrinsic constant LTL neural dynamics $(a_x^{i,j}, a_y^{i,j})$ completes the desired affine-linear transformation. The read-out is an updated encoded machine configuration, which is then synaptically projected back to the MCL units (c_x, c_y), initiating the next computation step (related to the original machine).

2.4. Neuronal Observation Models

In order to compare connectionist simulation results with experimental evidence from neurophysiology or psychology, one needs a mapping from the high-dimensional neural activation space $\Gamma \subset \mathbb{R}^n$ into a much lower-dimensional *observation space* that is spanned by $p \in \mathbb{N}$ observables $\varphi_k : \Gamma \to \mathbb{R}$ $(1 \leq k \leq p)$. A standard method for such a projection is PCA (Elman, 1991). If PCA is restricted to the first principal axis, the resulting scalar variable could be conceived as a measure of the overall activity in the neural network (as in beim Graben et al., 2008). Other important scalar observables that have been discussed in the literature are Smolensky's harmony (Smolensky, 1986)

$$H = \sum_{ij} u_i w_{ij} u_j$$

with $\boldsymbol{u} = (u_i)$ as the network's activation vector and $\boldsymbol{W} = (w_{ij})$ its synaptic weight matrix, or Amari's mean network activity (Amari, 1974)

$$A = \frac{1}{n} \sum_i u_i \,. \tag{32}$$

The development of biophysically inspired observation models is an important research field in computational neuroscience (beim Graben and Rodrigues, 2013) as it could eventually lead to "synthetic" local field potentials (LFPs), electroencephalogram (EEG), or event-related brain potentials (ERPs) (Barrès et al., 2013). We shall use Amari's measure (32) to derive such synthetic ERPs in what follows.

3. Results

The implementation of the R-ANN discussed in the previous Sections simulates a NDA in real-time and thus simulates its associated machine in real-time. More formally, it can be shown that under the map $\rho(\cdot)$ the commutativity property, $\zeta \circ \rho = \rho \circ \Phi$ (see commutative diagram of Figure 1) is satisfied. The NDA simulation (and thus the machine simulation) by the R-ANN is achieved by a combination of synaptic and neural computation among three neural types (MCL, BSL, and LTL) and with a total of neural units equal to

$$n_{\text{units}} = 2 + n_\alpha + n_\beta + 2n_\alpha n_\beta + 1 \tag{33}$$

where n_α and n_β are the number of sub-sequences that can appear respectively in the left and right Domain of Dependence of the VS from which the NDA and the R-ANN are constructed. That is, a total of 2 MCL units, $(n_\alpha + n_\beta)$ BSL units, $2n_\alpha n_\beta$ LTL units and a bias unit, that establish synaptic connections according to a synaptic weight matrix $\mathcal{A}$ of size $(n_{\text{units}} \times n_{\text{units}})$ following the connectivity pattern described in Figure 4. Specifically, the synaptic weights in $\mathcal{A}$ are entries from the set $\{0, 1, \frac{h}{2}, \frac{-h}{2}\} \cup \{a_k^{i,j} - h \mid i = 1, \ldots, n_\alpha n_\beta \ , \ j = 1, \ldots, n_\beta \ , \ k = x, y\}$, with the second set being the set of biases. A point worth mentioning is that the original formulation of the NDA relied on a simple Gödel encoding of the machine configurations, but subsequent work highlighted the advantages of using a more flexible representation by employing Cylinder sets, in order to preserve important structural relationships of the symbolic descriptions and to facilitate modeling (beim Graben and Potthast, 2009; beim Graben et al., 2008, 2004). Our R-ANN can be extended to incorporate a Cylinder set encoding of machine configurations by simply doubling the MCL and LTL layer.

An important modeling issue to consider is that of the halting conditions for the ANN, i.e. when to consider the computation as terminated. VSs, on which NDA and consequently our ANN model depend, do not define explicit halting conditions. However, two equally reasonable choices of halting conditions could be employed as follows. The first one is that of using a *homunculus* (beim Graben et al., 2004), an external observer which decides to intervene on the computation once some condition is met (for example, halting the computation when the input is in a certain region of the unit square). The second one is that of using a fixed point condition: implementing a machine halting state as an Identity branch on the NDA. This way a halting configuration will result in a fixed point on the NDA, and thus on

the R-ANN. In other words, the network's computation halts if and only if

$$\zeta_1(x', y') = (x', y'). \tag{34}$$

A halting by *homunculus* could be more appropriate in the context of interactive computation (beim Graben et al., 2008; Wegner, 1998) where constant and non-terminating interaction with the environment is assumed, or in cognitive modeling, where different kinds of fixed points, either desired or unwanted ones, are required in order to describe sequential decision problems (Rabinovich et al., 2008), such as linguistic garden paths (beim Graben et al., 2004, 2008).

We will now present two examples to demonstrate the strength of our developed methodology in mapping automata computation to R-ANN computation in real-time (an additional example on Turing Machines is available in the supplementary materials). The source code for all the examples is freely accessible via Carmantini (2015).

3.1. Example 1: Finite-State Locomotive Pattern Generator

FSMs are at the basis of many state-of-the-art approaches to the construction of locomotion controllers for articulated robots (see for example Alvarez-Alvarez et al., 2012; Collins and Ruina, 2005). They are easy to design, implement, and debug, and their relation with animal gait is well characterized (McGhee, 1968). On the other hand, recent research in robot locomotion control shows an increasing interest towards alternative approaches based on CPGs, neural networks capable of producing rhythmic patterns of activation in absence of rhythmic input sources. In his 2008 paper, Ijspeert presented the benefits and drawbacks of CPGs with respect to other approaches for robot locomotion control. We briefly summarize the benefits identified by the author: i) the rhythmic behavior supported by CPGs is robust to the transient perturbation of state variables; ii) CPGs are well-suited for distributed implementations (such as in modular robots); iii) CPGs reduce the dimensionality of the control problem by introducing few high-level control parameters allowing for the modulation of the locomotion; iv) CPGs are ideally suited for the integration of sensory feedback through coupling terms in the differential equations of the controller; v) CPGs often work well with learning and optimization algorithms. On the other hand, as specified by the author, CPG-based approaches are still lacking of a sound design methodology and theoretical grounding for their description. In the example

presented in this Section, we will show how our mapping could aid the design of CPGs producing arbitrary patterns for locomotion in robots, starting from a FSM description of the desired rhythmic pattern. By combining the two approaches, the design of these controllers benefits from the solid theoretical grounding of FSM-based locomotion and from its ease of design and implementation. To contextualize our derived CPG in terms of familiar animal locomotion, we qualitatively model the results of a well-known experiment on cat gait.

In their seminal work, Shik et al. (1966) applied different levels of electrical stimulation to the midbrain of a decerebrated cat. The authors observed transitions in the gait of the animal as an increasing level of stimulation was applied, eliciting first a *walk*, then a *trot* and finally a *gallop* gait. Our theoretical framework can qualitatively reproduce these experimental observations, by deriving a R-ANN which generates the relevant gait patterns, and reproduces the transition between them as a function of the applied stimulus strength. To keep the exposition simple, we will only consider the *walk* and *gallop* gaits, and the transition between the two. In the study of the mammalian quadruped gait, the four legs are numbered so that each gait can be associated with a certain sequence, given by the order in which the legs touch the ground over one gait cycle. The left and right hind legs are associated respectively with the numbers 1 and 2, and the left and right fore legs are associated respectively with the numbers 3 and 4. The gait cycle is assumed to start when the left hind leg touches the ground. A *walk* gait is thus defined by the sequence $(1, 3, 2, 4)$, and a *gallop* gait is defined by the sequence $(1, 2, 3, 4)$. At a very high level, the computation carried out by the CPG in charge of producing the gait patterns in the quadruped mammalian can be informally stated as: if stimulation from midbrain is low, sequentially activate legs following pattern $(1, 3, 2, 4)$. If it is high, sequentially activate legs from pattern $(1, 2, 3, 4)$. We can implement the low level and high level of stimulation as the two input symbols of a FSM, and construct the δ transition function to sequentially reproduce the two patterns by switching between states. The FSM can thus be defined as in Table 2.

This FSM can now be mapped (via our proposed approach) into a R-ANN, consisting in this case of 22 neural units (according to Equation 33). The chosen gamma functions for the Gödel encoding of this FSM are defined as

Symbols	States			
	q_1	q_2	q_3	q_4
<lo>	q_3	q_4	q_2	q_1
<hi>	q_2	q_3	q_4	q_1

Table 2: **State transition table for the simulated Central Pattern Generator finite-state automaton.** It is possible to observe how different input leads to different produced patterns, implemented as sequences of states.

follows:

$$\gamma_s(\sigma) := \begin{cases} 0 & \text{if } \sigma = \texttt{<lo>} \\ 1 & \text{if } \sigma = \texttt{<hi>} \end{cases} \qquad \gamma_q(q) := \begin{cases} 0 & \text{if } q = q_1 \\ 1 & \text{if } q = q_2 \\ 2 & \text{if } q = q_3 \\ 3 & \text{if } q = q_4 \end{cases}$$

The step-by-step dynamics of the derived R-ANN can be observed in Figure 7. Here we use the machine's input as the substrate for the external stimulus, which is ultimately encoded by the neural unit c_y within our R-ANN as shown in the bottom plot of Figure 7. Note how we manipulate the activation of c_y to gradually increase from a low to a high level of stimulation. That is, we introduce a continuous control parameter into an originally pure symbolic model, enabling us to carry out a bifurcation study in analogy with traditional coupled oscillator models (Golubitsky et al., 1999, 1998; Schöner et al., 1990; Collins and Richmond, 1994). Under this stimulation, the R-ANN defined by the mapping qualitatively reproduces the key features of the CPG involved in the locomotion and transitions described in Shik et al. (1966). In particular, it is possible to observe how low levels of stimulation elicit the production of the *walk* gait cycle, whereas an increase in the level of stimulation induces a sudden transition to the *gallop* gait cycle.

This key relation between the stimulation level (i.e a real control parameter) and the computation carried out by the network, which can be related to the underlying symbolic space thanks to the mapping, depends upon an informed decision in the gamma numbering of the states for the Gödel encoding. In fact, the chosen gamma numbering ensures that the unit square encoding of machine configurations where <lo> is the current input symbol

corresponds to all points (x, y) such that $x < \psi_y(\texttt{<hi>})$ where ψ_y is defined as in Equation 18, and specifically $\psi_y(\texttt{<hi>}) = \gamma_s(\texttt{<hi>})g_s^{-1} = \frac{1}{2}$. In terms of the underlying NDA representation, increasing the activation of c_y until its value reaches and exceeds $\frac{1}{2}$ corresponds to forcing the encoded machine state to cross the boundary between cells associated to a `<lo>` input symbol to those associated to a `<hi>` input symbol, thus causing a transition between a *walk* and a *gallop* gait. Note that in this example, we do not model halting conditions for the derived network, as it is not clear what halting means in the context of the computation performed by CPGs.

To summarize, we derived a CPG from a FSM description of a locomotion controller, inspired by results on the generation of gait patterns in the cat midbrain. By doing so, we outlined a new design methodology for CPG-based locomotion control in robots which does not suffer from some of the drawbacks of other CPG approaches, by grounding the description and design of the CPG on the theoretical grounding of FSM-based approaches. Some problematic aspects of the methodology we outlined are due to the discrete-time nature of our mapping. In fact, fully realizing the benefits of CPG-based approaches summarized at the beginning of this Section requires continuous time models. This notwithstanding, we believe that the proof of concept we provide here already shows encouraging results for future developments.

As an additional remark, the methods we describe in this paper are ideally suited for the deriving of neural networks implementing paradigms of interactive computation, as we will demonstrate shortly. This is especially relevant for the design of CPGs. In fact, recent research has unveiled a surprising degree of hierarchical organization in mammalian respiratory CPGs, which allows for a highly robust and flexible pattern production that can adapt to a variety of conditions (see for example work by Smith et al., 2013, 2007). Our methodology easily accommodates the mapping of hierarchies of automata to hierarchically organized neural networks, as we demonstrate in the next example through the modeling of garden-path parsing, a concept employed in language processing (beim Graben et al., 2004). Importantly, networks of automata could be used to design complex pattern generation in modular robots (see Spröwitz et al., 2014 for a recent example of modular robots using a distributed CPG for locomotion).

3.2. Example 2: Interactive Automata Networks

Interactive computation (Wegner, 1998) is a recent theoretical development that seeks to formalize the complexity of interactions that we observe in

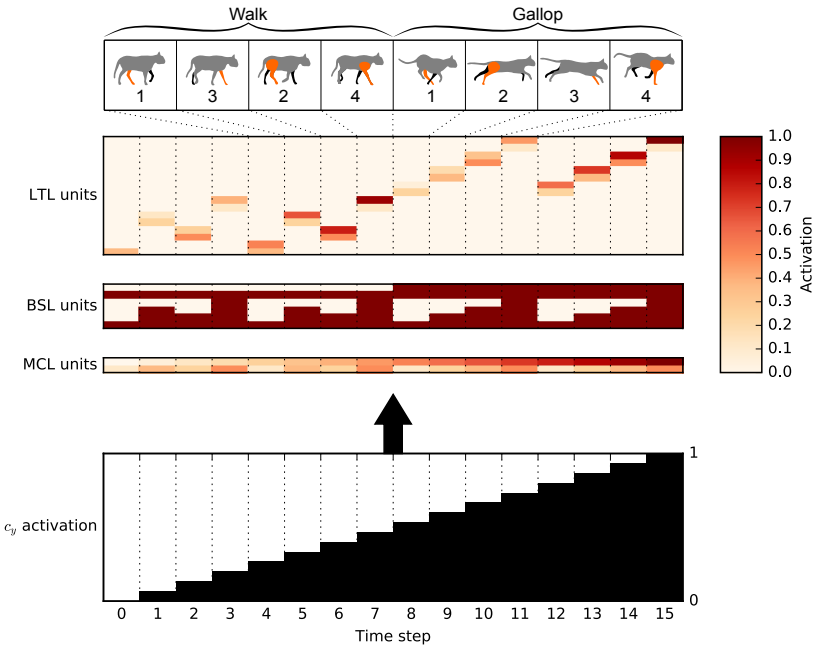


Figure 7: **Recurrent artificial neural network functioning as a Central Pattern Generator.** The network reproduces the qualitative behavior of the locomotive Central Pattern Generator described in Shik et al. (1966). In the bottom plot, the level of stimulation applied to the network through neuron c_y is shown. In the top three plots, the levels of activation of each neural unit in the three layers is shown for each time step. Note how two different patterns, *walk* and *gallop*, are generated depending on the level of stimulation. This results from the way the original finite-state machine was programmed.

real-world computing. In classical Automata Theory, the interaction between an automaton and the external world is restricted to an input-output relation. That is, the external world provides an input, the automaton performs its computation on that input, and then returns an output to the external world. Within the framework of interactive computation, instead, automata can interact with the external world (and with other automata) at every step of their computation. External forces can act on the configuration of the automaton, and the configuration can itself affect the external world. Clearly, this framework provides a much richer language to describe models of computation, and is especially useful to express notions of compositionality and concurrency. These constructs are essential not only in the study of modern computing systems, but also in the context of cognitive modeling. In this example, we will build a model of the human processing of locally ambiguous sentences by constructing a network of interactive automata. Through this proof-of-concept, we want to demonstrate the flexibility of our approach by showing how it can be seamlessly used to construct neural networks implementing interactive systems. In order to do so, we choose a system that i) is simple enough to allow for clear exposition, but complex enough to carry out a meaningful computation; ii) is composed by a range of different automata; iii) incorporates different forms of interaction between its automata components.

Garden-path sentences are locally ambiguous sentences that induce the temporary production of an erroneous parse by the reader, which is then forced to reconsider their interpretation of the previously presented material in order to finally reach a correct parse. Consider for example the sentence "I convinced her children are noisy". In reading the sentence, the reader first constructs an intermediate parse where "her children" is the object of the phrase "I convinced". After reading the rest of the sentence, the reader realizes that the intermediate parse was incorrect: "her" is the object of "I convinced", and "children are noisy" is a subordinate clause. The reader thus reanalyzes the sentence to produce a correct parse. Osterhout et al. (1994) have shown that the reanalysis of a sentence due to a garden-path is associated in the brain of the reader with a positive deflection 600 milliseconds (P600) after the onset of a garden-path – the word "are" in the example above – in sequentially presented sentences, as measured by a trial averaged electroencephalogram (thus obtaining event-related brain potentials).

Many proposals have been advanced to account for the mechanisms underlying the reanalysis of incorrectly parsed sentences due to garden-path

effects. In our model, we implement the reanalysis through a diagnosis and repair mechanism, described in Lewis (1998). By this account, the parser tries to incrementally build a parse as the sentence material is presented. If a dead-end is reached (i.e. the parser becomes stuck in a garden-path), the parser diagnoses the need for reanalysis, and the search space of possible continuations of the parse is modified by some repair operator that "bridges" the dead-end to another point in the search space, allowing the parser to correctly complete the processing of the sentence. The parser model we create implements this mechanism to process garden-path sentences where the local ambiguity is given by the incorrect assignment of the subject and object grammatical constituents.

In many languages, native speakers have been shown to prefer to interpret an ambiguous nominal constituent as a subject rather than an object. Consider for example the following two sentences, extracted from the ERP study on ambiguous pronouns by Frisch et al. (2004) on German speakers. Both sentences start with

Nachdem	*die Kommissarin*	*den Detektiv*	*getroffen hatte*	...
After	the cop	the detective	had met	...

"After the cop had met the detective, ..."

One of the sentences then continues with a clause in subject-object order (s-o sentence), i.e. the preferred order in the parsing of ambiguous constituents:

(s-o sentence)

...	*sah*	*sie*$_{\mathbf{s}}$	*den Schmuggler*$_{\mathbf{o}}$
...	saw	she	the smuggler

... "she saw the smuggler"

In this case, the reader correctly interprets "sie" to be the subject of the second clause, and "den Schmuggler" as the object (as "den Schmuggler" is in the accusative case, thus specifying a direct object to the verb "sah").
The second sentence is instead in the dispreferred object-subject order (o-s sentence):

(o-s sentence)

...	*sah*	*sie*$_{\mathbf{o}}$	*der Schmuggler*$_{\mathbf{s}}$
...	saw	she	the smuggler

... "the smuggler saw her"

The psycholinguistic study by Frisch et al. (2004) has shown that the reader first tries to apply the preferred subject-object parsing strategy to this clause (and sentences with similar subject/object pronoun ambiguity). The reader thus initially interprets "sie" as the subject of the clause in nominative case, expecting it to be followed by the object in accusative. Upon further reading, however, they realize that "der Schmuggler" is in the nominative case instead, and thus has to be the subject. This leads the reader to reconsider the previous material to correctly parse "sie" as a pronoun in accusative case, the direct object of the verb "sah". This reanalysis was observed as a P600 effect in the ERP.
At a high level of abstraction (beim Graben et al., 2004), we can capture the structure of these sentences through a CFG G with production rules:

$$\mathtt{S} \rightarrow \mathtt{s}\ \mathtt{o} \qquad (s\text{-}o)$$
$$\mathtt{S} \rightarrow \mathtt{o}\ \mathtt{s}, \qquad (o\text{-}s)$$

where `S` is a distinguished starting non-terminal, and where the `s` and `o` terminals stand respectively for "subject" and "object" phrase.

In our model we thus split the G grammar into two grammars $G_{s\text{-}o}$ and $G_{o\text{-}s}$, comprising respectively of the *s-o* and *o-s* production rules (beim Graben et al., 2004), and reflecting the existence of two strategies in the parsing of sentences with subject/object pronoun ambiguity. To recognize the two different sentence structures, our model is endowed with two specialized TDRs, constructed from the $G_{s\text{-}o}$ and $G_{o\text{-}s}$ grammars as shown in section 2.1.2. Initially, the *s-o* TDR is tried on the input, to model the subject-object interpretation preference. In case it fails because of a garden path, the model acts as prescribed by a diagnosis and repair account. That is, it first diagnoses that a problem has arisen in parsing, repairs the parse, and finally switches strategy to correctly parse the input. In order to implement the diagnosis step, our model needs a way to monitor the state of the parse and extract the relevant diagnostic information. We implement this through a *Diagnosis* PDA (see Table 3), which compares the current parse with that from the previous time step; if the parse didn't change, that means that the parser is stuck and can't process the input further. In that case the *Diagnosis* PDA changes its state to an "error" state, thus implementing the diagnosis step. The repair step is realized by introducing a *Repair* VS , that can be described by the following rewriting rule:

$$\mathtt{s}\ \mathtt{o}\ .\ w \rightarrow \mathtt{o}\ \mathtt{s}\ .\ w, \qquad (35)$$

Symbols	States		
	${}^{\text{pda}}q_{\text{idle}}$	${}^{\text{pda}}q_{\text{parsing}}$	${}^{\text{pda}}q_{\text{error}}$
$(\sqcup, \sqcup)$	$({}^{\text{pda}}q_{\text{idle}}\ ,\ \sqcup)$	$({}^{\text{pda}}q_{\text{idle}}\ ,\ \sqcup)$	$({}^{\text{pda}}q_{\text{idle}}\ ,\ \sqcup)$
(x, y)	$({}^{\text{pda}}q_{\text{parsing}}\ ,\ y)$	$({}^{\text{pda}}q_{\text{parsing}}\ ,\ y)$	$({}^{\text{pda}}q_{\text{parsing}}\ ,\ y)$
$(x, x);\ x \neq \sqcup$	$({}^{\text{pda}}q_{\text{error}}\ ,\ x)$	$({}^{\text{pda}}q_{\text{error}}\ ,\ x)$	$({}^{\text{pda}}q_{\text{error}}\ ,\ x)$

Table 3: **State transition table for the *Diagnosis* push-down automaton (PDA).** The input to this machine is the parse produced by the top-down recognizers (TDRs). For any state, input symbol and stack symbol, the machine pushes its input to the stack, in order to be able to compare its current input with the one from the previous time step. In particular, if the input symbol and top-of-stack are blank symbols, the machine transitions to an "idle" state, signaling that nothing is happening; if the current input and the one from the previous time step are different, the machine transitions to a "parsing" state, signaling that the TDRs are successfully parsing their input; if the current input and the one from the previous time step are the same (but not both blanks), then the TDR parsing the input is stuck, and the machine transitions to an "error" state.

corresponding to a reanalysis of the ambiguous sentence in terms of the dispreferred object-subject sentence structure. Once the sentence has been reanalyzed and thus the parse repaired, the second parser can proceed to process the input until it has been completely consumed and the stack is emptied. In order to switch strategies, our model needs a higher-level controller that has access to diagnostic information about the current parse, and decides which parsing strategy to apply. In particular, this controller should first activate the preferred *s-o* TDR. If the parser failed (as signaled by the *Diagnosis* PDA) then the higher-level controller should first activate the *Repair* VS to allow for the reanalysis of the ambiguous sentence, and subsequently activate the *o-s* TDR. We implement the high level controller through a *Strategy* FSM (see Table 4), endowed with the capability of selectively activating the *s-o* and *o-s* TDRs, as well as the *Repair* VS, by switching its internal state. This machine receives the diagnostic information provided by the *Diagnosis* PDA as input. The FSM has three states, namely an "s-o" state, a "repair" state, and an "o-s" state. By switching between these states, the FSM can activate the respective automata. Note that this form of interaction is not defined for the VS introduced in Section 2.1.1. That is, we do not define a way for a VS to "call" other shifts. Extending VSs to incorporate notions

Symbols	States		
	${}^{\text{fsm}}q_{\text{s-o}}$	${}^{\text{fsm}}q_{\text{o-s}}$	${}^{\text{fsm}}q_{\text{repair}}$
${}^{\text{pda}}q_{\text{idle}}$	${}^{\text{fsm}}q_{\text{s-o}}$	${}^{\text{fsm}}q_{\text{s-o}}$	${}^{\text{fsm}}q_{\text{s-o}}$
${}^{\text{pda}}q_{\text{parsing}}$	${}^{\text{fsm}}q_{\text{s-o}}$	${}^{\text{fsm}}q_{\text{o-s}}$	${}^{\text{fsm}}q_{\text{o-s}}$
${}^{\text{pda}}q_{\text{error}}$	${}^{\text{fsm}}q_{\text{repair}}$	${}^{\text{fsm}}q_{\text{o-s}}$	${}^{\text{fsm}}q_{\text{o-s}}$

Table 4: **State transition table for the *Strategy* finite-state machine (FSM).** The input to this machine is the diagnostic information produced by the *Diagnosis* push-down automaton (PDA), i.e. its state. The FSM starts in state ${}^{\text{fsm}}q_{\text{s-o}}$. In fact, the preferred parsing strategy is that implemented by the *s-o* top-down recognizer (TDR), corresponding to the parsing of subject-object sentences, so that it is tried first. If the *s-o* TDR fails, the *Diagnosis* PDA signals an error; the input sentence is not in subject-object order, and a switch of parsing strategy is needed. The *Strategy* FSM first changes state to ${}^{\text{fsm}}q_{\text{repair}}$, activating the *Repair* versatile shift (VS) so that the switch can take place. Repairing the parse leads the *Diagnosis* PDA to signal that the parsing started again, so that the new input for the *Strategy* FSM becomes again ${}^{\text{pda}}q_{\text{parsing}}$. Given ${}^{\text{pda}}q_{\text{parsing}}$ in input and ${}^{\text{fsm}}q_{\text{repair}}$ as a current state, the FSM moves to the ${}^{\text{fsm}}q_{\text{o-s}}$ state, leading to the activation of the *o-s* TDR, until the input has been parsed.

of compositionality and concurrency will allow the refining of the mapping presented in this paper to reflect these new capabilities. For the moment, we just want to demonstrate the possibilities opened by the present work; for this reason, we will implement the "subroutine" capability in our neural network through a familiar mechanism already encountered in the previous Sections, ignoring momentarily the missing theoretical details and leaving their definition for future work.

To avoid race conditions, at most one automaton in the interactive network can re-write symbols in a sub-sequence at any given computation step . The "parse" sub-sequence can only be read, but not re-written, by the *Diagnosis* PDA. Similarly, the "diagnosis" sub-sequence can only be read, but not re-written, by the *Strategy* FSM. Furthermore, the selective activation of the *s-o* TDR, the *o-s* TDR, and the *Repair* VS operated by the *Strategy* FSM ensures that at any given computation step only one between these automata can perform symbolic re-writing on the "input" and "parse" sub-sequences.

To map the system of interactive automata to a R-ANN, we first convert each of its component in the familiar way, as described in the previous Sections. That is, the *s-o* and *o-s* TDRs, the *Repair* VS, the *Diagnosis*

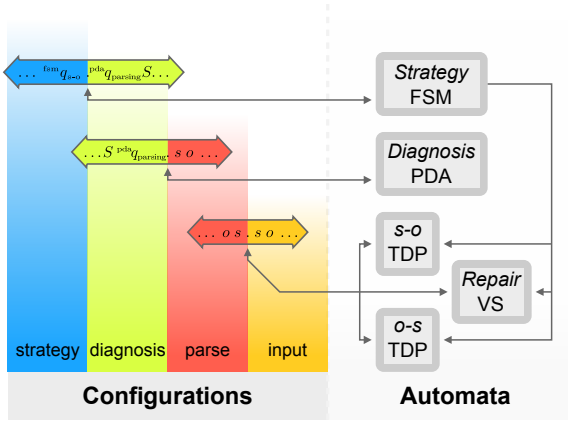


Figure 8: **Interactive automata network for parsing of garden path sentences.** The figure shows the complete system described in Section 3.2. For simplicity, we show the various automata components as acting on their configurations represented as dotted sequences. Dotted sub-sequences of the same color are coupled, i.e. they are for all intents and purposes the same sub-sequence. For example, the "parse" dotted sub-sequence that contains the current stack of the top-down recognizers (TDRs) and of the *Repair* versatile shift (VS), is at the same time the input tape of the *Diagnosis* push-down automaton (PDA). Similarly, the "diagnosis" sub-sequence that stores the current state and stack of the *Diagnosis* PDA, is at the same time the input tape of the *Strategy* finite-state machine (FSM). Note that a second form of interaction, other than that allowed through the sharing of dotted sequences, is present in the automaton. In particular, the *s-o* and *o-s* TDRs and the *Repair* VS are activated based on the state of the *Strategy* FSM.

PDA, and the *Strategy* FSM are first converted to VSs acting on dotted sequences, then mapped to their NDA representation and finally to R-ANNs. The Gödelizations of the "input", "parse" and "strategy" sub-sequences are defined as in Equation 18, with each of the gamma enumerating functions defined as follows:

$$\begin{aligned} \gamma_{\text{input}} &:= \{(\sqcup, 0), (\mathtt{S}, 1), (\mathtt{o}, 2), (\mathtt{s}, 3)\} \\ \gamma_{\text{parse}} &:= \{(\sqcup, 0), (\mathtt{o}, 1), (\mathtt{s}, 2)\} \\ \gamma_{\text{diagnosis}} &:= \{({}^{\text{pda}}q_{\text{idle}}, 0), ({}^{\text{pda}}q_{\text{parsing}}, 1), ({}^{\text{pda}}q_{\text{error}}, 2)\} \end{aligned} \tag{36}$$

where each function is represented as a set of (σ, k) pairs, with σ being a symbol and $k \in \mathbb{N}$ its enumeration. The Gödelization of the "diagnosis" sub-sequence is instead defined as in Equation 19, with

$$\gamma_{\text{strategy}} := \{({}^{\text{fsm}}q_{\text{s-o}}, 0), ({}^{\text{fsm}}q_{\text{o-s}}, 1), ({}^{\text{fsm}}q_{\text{repair}}, 2)\}$$

enumerating the states of the *Diagnosis* PDA, and γ_{parse} (already defined in Equation 36) enumerating its stack symbols. Having mapped each of the machines to a R-ANN, we can use the derived networks as components of the overall system architecture (see Figure 9 for the full architecture). In order to simplify the exposition, we construct the overall network to feature only one set of recurrent connections. To do so, we endow our architecture with 4 Configuration Layers (CLs), containing the "strategy", "diagnosis", "parse", and "input" sub-sequences. Between each CL and the next, the network components derived from the automata are connected to perform their part of the processing on the relevant subsequences. In particular, if the VS representation of an automaton acts on some $\alpha.\beta$ dotted sequence, the input of its associated network component is connected to the units encoding the α and β subsequences in the i-th CL, whereas its output (which is a recurrent connection to the MCL layer in the original mapping) is connected to the units encoding α and β in the $(i+1)$-th CL. The final CL is connected with a $\mathcal{I}_{4\times4}$ synaptic weight matrix to the first CL layer (i.e. each unit encoding a subsequence of the last CL is connected with a weight of 1 to the same unit in the first CL). Finally, to implement the subroutine call capabilities of the strategy FSM, we add a *Meta* branch selection layer that takes the "strategy" subsequence as input, and is connected with the lateral inhibition connection pattern specified in Section 2.3.3 to the *s-o* and *o-s* TDRs, and to the *Repair* VS. Note how this creates a nested structure, with the *s-o* TDR, the *o-s* TDR, and the *Repair* VS functioning as higher-level symbolic

operations of a *Parser* machine. This is reflected in the nested structure of the *Parser* R-ANN sub-network, where the lower-level machines function as cells in a LTL, controlled by the *Meta* BSL (see Figure 9).

In Figure 10 we show the network activation when two different sentence structures are presented in input. In particular, note the serial activation of the *s-o* TDR, *Repair* VS and *o-s* TDR sub-networks when a object-subject sentence is presented. By mapping the parser from a machine evolving in a symbolic space to a neural network evolving in a vectorial space, we are now able to compute synthetic event-related potentials, or "synth-ERPs", (beim Graben et al., 2008; Barrès et al., 2013) as trial-averages of the mean network activation, as discussed in Figure 11. This is achieved by calculating the mean global network activation according to Amari (1974) (Equation 32) for a simulation over 100 trials for each input stimulus, where random initial conditions compatible with the symbologram representation of the input are prepared according to beim Graben et al. (2008). In brief, symbologram-compatible random initial conditions are generated through the Gödelization of sequences of the form $w_\alpha u.v w_\beta$, where $u.v$ is the dotted sequence describing the input to the system, and $w_\alpha, w_\beta \in \mathbf{A}^*$ are random sequences of symbols in $\mathbf{A}$.

As Figure 11 reveals, the network shows a P600-like effect in the processing of garden-path sentences, with a peak of increased and sustained activation with respect to the control condition. The simplified model of garden-path processing we presented here does not yet allow for a direct quantitative comparison with experiments such as in Frisch et al. (2004) (in fact, a carefully crafted model would require a level of detail and attention which goes beyond the scope of this paper). Yet, these simulations could be the starting point for more detailed statistical correlation analyses (beim Graben and Drenhaus, 2012; Frank et al., 2015) in future work, relating these computations to electrophysiological measurements.

4. Discussion and Outlook

In this study we have developed a constructive, transparent, modular and parsimonious mapping from symbolic algorithms to neural networks. We first introduced a novel shift map, the versatile shift, that extends the generalized shift and allows for the real-time simulation of a range of symbolic models of computation. We then showed how VSs can be represented on a vectorial space through Gödelization, obtaining piecewise affine-linear systems

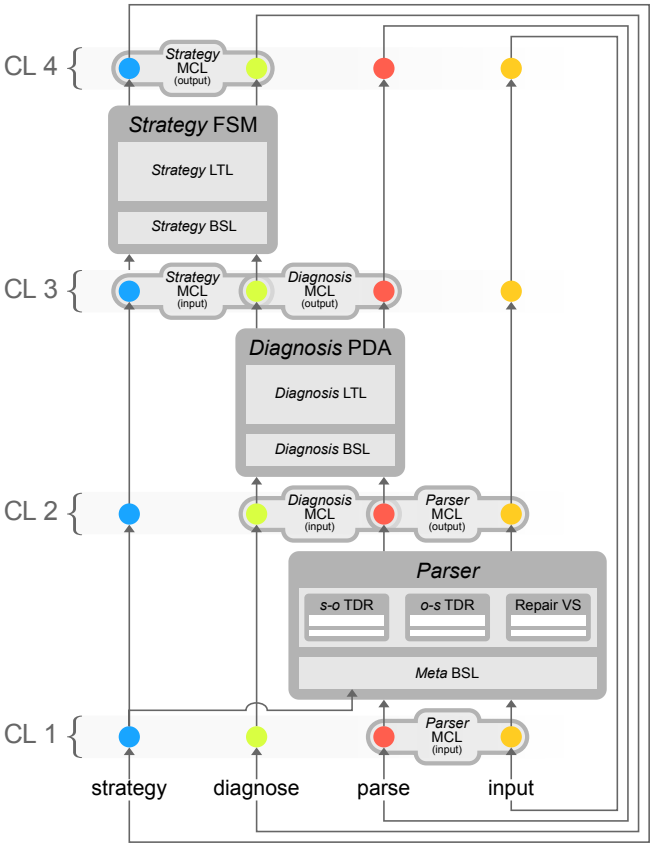


Figure 9: **Garden-path parsing network architecture.** In order to simplify the exposition, we construct our network such that the only recurrent connection is that from the last to the first layer of the network (i.e. CL 4 to CL 1, where CL stands for Configuration Layer). Note that the *Parser* sub-network is itself composed of the *s-o* top-down recognizer (TDR), the *o-s* TDR, and the *Repair* versatile shift (VS) sub-networks. These are arranged as cells of a linear transformation layer (LTL), and selectively activated by a *Meta* branch selection layer (BSL) controlled by a "strategy" neural unit.

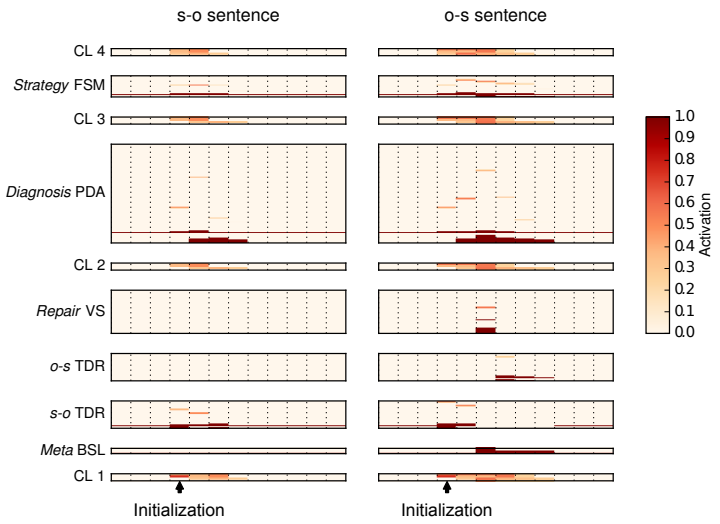


Figure 10: **Network activation for subject-object and object-subject sentence presentation.** Notice the serial activation of the *s-o*, *Repair* and *o-s* sub-networks in case of an object-subject sentence presentation, and the longer "tail" of activation, reflecting the additional computation needed to process the dispreferred input.

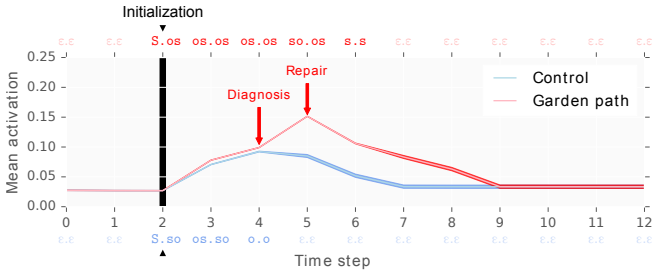


Figure 11: **Synthetic P600 event-related brain potential as mean Network activation for random cloud of initial conditions.** In this figure we show the mean global network activation calculated through Equation 32 for each time step of two simulations, averaged over 100 trials. For each of the two simulations, we run the network presenting at time $t = 2$ one of 100 random inputs generated compatibly to the symbologram representation of one of two sequences. In other words, noise is added to each input such that, if the input was generated by Gödelizing a sequence of length n, decoding the input would yield the original sequence in the first n symbols, with the rest being a random symbolic continuation. If stronger noise was added instead, that would have prevented the network to correctly perform its computation, as we would have destroyed essential input information. In blue, we show the averaged mean activation (light blue) and its standard deviation (dark blue) for a presented input encoding the sequence `S.so`, representing an input sequence in subject-object order, i.e. the network's preferred order as explained in Section 3.2. Note that the parsing is completed at $t = 5$. In red, the averaged mean activation (light red) and its standard deviation (dark red) for an input encoding the sequence `S.os`, representing an input sequence in object-subject order, leading to a garden path in the parsing of the input. The time at which the diagnosis ($t = 4$) and repair ($t = 5$) steps are carried out in the symbolic interactive system (and thus in its recurrent artificial neural network mapping) is indicated by arrows. We also report, at the top and bottom of the plot, the configuration of the parser networks as a dotted sequence for each time step, for respectively the garden path and the control condition. Note how the garden-path processing is associated with a strong divergence in activation starting from time $t = 5$, and followed by a longer tail than that of the network in the control (preferred) condition. This reflects the additional computation needed by the network to successfully resolve the garden path in parsing, and qualitatively corresponds to the P600 event-related brain potential measured in psycholinguistics experiments (see Section 3.2). Furthermore, note that in both conditions the network starts and returns to a "resting state", waiting for input to process from the external world, implementing a notion of continuous computation which is the hallmark of interactive systems.

on the unit square known as nonlinear dynamical automata (Tabor, 2000; Tabor et al., 2013; beim Graben et al., 2004, 2008). Finally, we presented a modular R-ANN architecture that simulates the dynamics of NDA. The proposed architecture consists of three layers: a machine configuration layer representing the NDA state, and thus the symbolic data in the simulated automaton; a branch selection layer implementing the NDA switching rule, thus characterizing the automaton's decision space, or control; and the linear transformation layer implementing the set of piecewise affine-linear functions in the NDA, i.e. the vectorial representation of the symbolic operations defined in the transition table of the simulated automaton. Additionally, the linear transformation layer is itself modular, in that each operation specified by the δ transition function of the simulated automaton is applied by a specific pair of units in the layer.

The mapping can be used to simulate any Turing machine through R-ANNs, thus making the architecture universal (an example of the mapping on Turing Machines is reported in the supplementary materials). In particular, it is possible to simulate the 7-states 4-symbols UTM by Minsky (1962) in real-time with a R-ANN consisting of 259 units [6] (see Equation 33), and the 6-states 4-symbols UTM by Neary and Woods (2009) with one consisting of 223 units.

It is important to analyze some of the modeling choices that have been made in the R-ANN architecture we described. A choice worth discussing is that of implementing biases as synaptic projections from an always-active unit as opposed to implementing them as parameters intrinsic to the individual units. We decided for simplicity to add a bias unit. Nonetheless, a parameterized bias would have been equally reasonable. While it does not have strong bearings on the model here discussed, it is interesting to note that the specific choice of implementation does more or less put the emphasis on a predominantly synaptic computation versus a computation which is more distributed between the synaptic and the neuron level, reflecting similar issues to be considered in the biological domain. A second consideration concerns the cell's boundaries in the NDA. In fact, the distance between the right bound of a cell and the left bound of the next one is zero. This poses some challenges, as even extremely small noise on the state vector at a

[6] This implies a reduction factor of 1/3 when compared to the solution by Siegelmann and Sontag (1991, 1995), which simulates Minsky's UTM with a network of 886 units.

boundary can lead to an erroneous application of the switching rule on the real state, and thus to a disruption of the computation. This is of course reflected in the dynamics of the associated R-ANN as well. Siegelmann and Sontag (1995) solve this issue by using a Cantor encoding as opposed to a simple Gödelization, ensuring a greater than zero distance between two encoded configurations with different leading symbols. The same methods can be applied here. Interestingly, by switching to a Cantor encoding, the Heaviside units in the BSL layer can be substituted with functionally equivalent Ramp units, so that the R-ANN would only make use of linear units à la Siegelmann and Sontag.

We will now first discuss the advantages of our approach over those based on eliminative connectionism, and then the advances that the present work brings to transparent connectionism.

Compared to eliminative approaches, our work allows the direct interpretation of the representations and the dynamics in the derived network in terms of symbolic computation. This has many important consequences. First, while conventional neural networks have to be trained on large data sets (usually using backpropagation or related algorithms, see Werbos, 1990) our method does not require any training, as the synaptic weight matrix is explicitly designed from the machine table of the encoded automaton. Emergent representations and operations are not opaquely encoded in several hidden layers but transparently realized through Gödelization of symbolic configurations. Second, even when considering learning applications – which we plan to explore in future developments – the derived approach could bring about the exciting possibility of a symbolic read-out of a learned algorithm from the network weights; Note that in this architecture all weights are necessarily fixed, with the exception of the connections encoding the symbolic operations in the simulated automaton, i.e. those between the MCL and the LTL layer. Third, anchoring the computation of the network to well-understood computation models is worthwhile when tackling problems that can benefit from the integration of the two perspectives. In the first example, we constructed a R-ANN (24 units) performing a FSM machine computation abstracting a CPG for animal locomotion. FSMs are widely used in locomotion controllers in robotics, because of their simplicity and strong theoretical grounding in relation to animal locomotion. On the other hand, neural implementations of CPG have many desirable characteristics (as discussed in Ijspeert, 2008) that are not present in FSM-based implementations, but they are difficult to engineer. We showed that by integrating the two approaches we can tackle

the problem of pattern generation in robotic locomotion more effectively. Of course, a satisfactory solution would entail the use of continuous-time models in the mapping; nevertheless, our preliminary results already present distinct benefits in the integration of the two approaches as compared with their use in isolation. Fourth, having a complete understanding of the network's inner workings allows for the intelligent manipulation of its parameters. In the discussed CPG example, understanding the computation carried out by the derived network allowed us to introduce a continuous control parameter eliciting a bifurcation in the dynamics of the network, as present in systems of coupled nonlinear oscillator models (Golubitsky et al., 1999, 1998; Schöner et al., 1990; Collins and Richmond, 1994), widely studied in the CPG literature.

In regards to previous work on transparent connectionism, our work advances the field in several ways. As a first advancement, by introducing VSs we are now able to use NDA to simulate a broad range of symbolic computation models in real-time, extending the original work by Moore (1990, 1991). Interestingly, it would be straightforward to define n-sided infinite dotted sequences (where the dot splits a sequence in its n one-sided infinite components), and extended VSs on these. By Gödelization, we would obtain NDA on the n-dimensional hypercube, which could be simulated by R-ANNs through a straightforward extension of the architecture presented in this work. This would further extend the range of real-time simulable computational models to automata with multiple tapes or stacks (Aho, 1969; Weir, 1994). Secondly, by basing our construction on NDA, we obtain an architecture characterized by a fully distributed representation coupled with a granular modularity, differentiating our approach from previous work and granting a series of advantages. The mapping is transparent not only with regards to the representations (the data), but also with regards to the symbolic operations defined in the simulated computational model and their control, all clearly localizable in the architecture. We regard this as an advance in itself (in line with the goals of transparent connectionism), but it also allows, for example, for the straightforward mapping of interactive automata networks to R-ANNs. This is of fundamental importance, as the framework of interactive computation provides a rich language for the description of many complex systems, for example in cognitive modeling. In the second example we constructed a network of interacting automata as a diagnosis and repair model (Lewis, 1998; beim Graben et al., 2004, 2008) for the reanalysis of linguistic garden path sentences. The network consisted of three PDA (two

of them as TDRs), a VS, and one FSM as a master control program, with each component carrying out a specific and intelligible task in the overall computation. We then mapped this network to a R-ANN (266 units), thus obtaining a symbolic/connectionist implementation of a cognitive model. Interestingly, due to the multiple levels of hierarchical organization that can be present in the automata network (which comprises nesting, as in the diagnosis and repair network) and, thus, in the derived R-ANN, one could even speculate about thermodynamic limit networks when the number of modules approaches infinity, presenting emergent scale-free or small world properties (Albert and Barabási, 2002). The granular modularity of our approach is also a key advancement when considering the possibility of correlational studies with neurophysiological measurements. In previous work we showed how to devise large-scale biophysical observation models in order to correlate top-down modeling approaches with neurophysiological data obtained from bottom-up measurements (Amari, 1974; beim Graben and Rodrigues, 2013). The process involves associating neural units of our model with neuronal masses (Lopes da Silva et al., 1974; Jansen and Rit, 1995) or Hebbian cell assemblies (Hebb, 1949; Wennekers and Palm, 2009; Huyck, 2009) in large-scale brain models, as investigated, e.g., in neural field theory. With this setup we then show that our observational models lead to improved interpretation, e.g of "synthetic event-related brain potentials" (as discussed in Section 2.4, see beim Graben et al., 2008; Barrès et al., 2013) as used in computational neurolinguistics studies (Gigley, 1985; beim Graben and Drenhaus, 2012; Barrès et al., 2013), where mental/cognitive states can be associated to metastable states of a dynamical system. In the second example presented here, we computed Amari's mean activation (Amari, 1974) as an observation model for the diagnose and repair R-ANN, in order to obtain synthetic ERPs (beim Graben et al., 2008; Barrès et al., 2013). Qualitatively, the computed signal exhibited a similar divergence between conditions as measured in the experiment presented in Frisch et al. (2004). While preliminary, these are already encouraging results for the development of our approach in this direction. In future work, we envisage that it will be possible to selectively correlate electrophysiological measurements with specific components in a derived R-ANN, as informed by a suitable symbolic model for the computation underlying the measured quantities. As a third point of interest, the architecture presents a clear 2D spatial organization in its layout, particularly at the level of LTL (as highlighted in Figure 6). In a NDA, different transformations are applied based on the position of the Gödelized automaton data on the unit square.

In the R-ANN architecture, this is implemented through the BSL, which performs a form of spatial pattern matching, activating a specific pair of units in the LTL through a lateral inhibition mechanism. When considering extensions to models of higher complexity, the functionality of BSL and LTL could be implemented through the use of a grid of units with receptive fields, as defined for example in self-organizing maps (SOMs, see Kohonen, 1982; Kohonen and Somervuo, 1998).

In future work, we plan to overcome fundamental issues with the current model which have bearing both in relation to learning applications and to the extension of the model to continuous dynamics. For what concerns the learning of algorithms from data, the current model suffers from a missing end-to-end differentiability, due to the use of Gödel encodings. This is a serious limitation, as it prevents the use of gradient descent methods for the training of the network's weights. Future work will have to address this limitation, possibly relying on methods of data access and manipulation akin to modern R-ANN approaches such as in Weston et al. (2014); Graves et al. (2014); Grefenstette et al. (2015); Joulin and Mikolov (2015); Sukhbaatar et al. (2015). Encouraging work on the learning of exponential state growth languages by Fractal Learning Neural Networks (Tabor, 2003, 2011) could also inform a revised trainable architecture.

With regards to the extension of the model to continuous dynamics, there are many ways in which this could be achieved in future work. Importantly, we are mostly interested in extensions to continuous-time models that are excitable. In such systems, trajectories can be perturbed away from a stable equilibrium (or rest state) and come back to it only after a large excursion (or spike) in the phase space, upon sufficiently strong input; biophysical examples of excitable models were initiated in Hodgkin and Huxley, 1952. One possibility would be to first extend the mapping to discrete-time excitable models (as in map-based neuronal models, see Ibarz et al., 2011; Girardi-Schappo et al., 2013), and then move to continuous time via so-called *suspension* procedures. There are some potential issues in this endeavor. First of all it would be crucial to first explore and understand the possible relationships between excitable regimes in neural models and symbolic dynamics in a computation. That is, to answer the question: how does the excitability property translate in the realm of symbolic computation? We think there could be meaningful answers to this question when tackled through the framework of interactive computation. Another potential issue is that the suspension process is non-unique and non-trivial in the general case; moreover, it does not guarantee

that the excitability property will be preserved.

Excitability is a crucial matter when dealing with neural tissue of lower brain structures, such as the Brain stem, where it is possible to neurophysiologically identify clear and small neuronal networks. However, neural networks models are not the most appropriate level of description for higher cortical structures, due to the presence of large and highly interconnected neuronal masses. Models of these structures express slow but large scale processes as measured by LFP/EEG. In this context, an alternative approach to achieve continuous-time dynamics, which we have already explored to some extent in previous work, is by the framework of *heteroclinic dynamics*, where Turing machine configurations can be interpreted as metastable states with attracting and repelling directions (beim Graben and Potthast, 2009; Tsuda, 2001; Rabinovich et al., 2008; Krupa, 1997), or by the framework of multiple-time scale dynamical systems (Desroches et al., 2013; Fernández-García et al., 2015).

Acknowledgements

This research has been supported by a Heisenberg fellowship (GR 3711/1-2) of the German Research Foundation (DFG) awarded to PbG.

References

Aho, A. V. (1969). Nested stack automata. *Journal of the Association for Computing Machinery*, 16(3):383 – 406.

Aho, A. V. and Ullman, J. D. (1972). *The Theory of Parsing, Translation and Compiling*. Prentice-Hall.

Albert, R. and Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47 – 97.

Alvarez-Alvarez, A., Trivino, G., and Cordón, O. (2012). Human gait modeling using a genetic fuzzy finite state machine. *IEEE Transactions on Fuzzy Systems*, 20(2):205–223.

Amari, S.-I. (1974). A method of statistical neurodynamics. *Kybernetik*, 14:201 – 215.

Barrès, V., III, A. S., and Arbib, M. (2013). Synthetic event-related potentials: A computational bridge between neurolinguistic models and experiments. *Neural Networks*, 37:66 – 92.

beim Graben, P. and Drenhaus, H. (2012). Computationelle Neurolinguistik. *Zeitschrift für Germanistische Linguistik*, 40(1):97 – 125.

beim Graben, P., Gerth, S., and Vasishth, S. (2008). Towards dynamical system models of language-related brain potentials. *Cognitive Neurodynamics*, 2(3):229–255.

beim Graben, P., Jurish, B., Saddy, D., and Frisch, S. (2004). Language processing by dynamical systems. *International Journal of Bifurcation and Chaos*, 14(02):599–621.

beim Graben, P. and Potthast, R. (2009). Inverse problems in dynamic cognitive modeling. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 19(1):015103.

beim Graben, P. and Rodrigues, S. (2013). A biophysical observation model for field potentials of networks of leaky integrate-and-fire neurons. *Frontiers in Computational Neuroscience*, 6(100).

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798 – 1828.

Blutner, R. (2011). Taking a broader view: Abstraction and idealization. *Theoretical Linguistics*, 37(1-2):27 – 35.

Cabessa, J. and Siegelmann, H. T. (2012). The computational power of interactive recurrent neural networks. *Neural Computation*, 24(4):996–1019.

Cabessa, J. and Villa, A. E. (2012). The expressive power of analog recurrent neural networks on infinite input streams. *Theoretical Computer Science*, 436:23–34.

Cabessa, J. and Villa, A. E. (2013). The super-turing computational power of interactive evolving recurrent neural networks. In *Artificial Neural Networks and Machine Learning–ICANN 2013*, pages 58–65. Springer.

Carmantini, G. S. (2015). Turing neural networks. *GitHub repository*. `https://github.com/TuringMachinegun/Turing_Neural_Networks`.

Carmantini, G. S., beim Graben, P., Desroches, M., and Rodrigues, S. (2015). Turing computation with recurrent artificial neural networks. In *Proceedings of the NIPS Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches*, pages 5 – 13. arXiv:1511.01427 [cs.NE].

Christiansen, M. H. and Chater, N. (1999). Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, 23(4):157 – 205.

Collins, J. J. and Richmond, S. A. (1994). Hard-wired central pattern generators for quadrupedal locomotion. *Biological Cybernetics*, 71(5):375 – 385.

Collins, S. H. and Ruina, A. (2005). A bipedal walking robot with efficient and human-like gait. In *ICRA 2005. Proceedings of the 2005 IEEE International Conference on Robotics and Automation.*, pages 1983–1988. IEEE.

Davis, M. D., Sigal, R., and Weyuker, E. J., editors (1994). *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science.* Academic Press, Harcourt, Brace and Company.

Desroches, M., Krupa, M., and Rodrigues, S. (2013). Inflection, canards and excitability threshold in neuronal models. *Journal of Mathematical Biology*, 67(4):989–1017.

Dominey, P. F. (1995). Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning. *Biological cybernetics*, 73(3):265–274.

Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., and Rasmussen, D. (2012). A large-scale model of the functioning brain. *Science*, 338(6111):1202 – 1205.

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

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

Elman, J. L. (1995). Language as a dynamical system. In Port, R. F. and van Gelder, T., editors, *Mind as Motion: Explorations in the Dynamics of Cognition*, pages 195 – 223. MIT Press, Cambridge (MA).

Farkas, I. and Crocker, M. W. (2008). Syntactic systematicity in sentence processing with a recurrent self-organizing network. *Neurocomputing*, 71:1172 – 1179.

Fernández-García, S., Desroches, M., Krupa, M., and Clément, F. (2015). A multiple time scale coupling of piecewise linear oscillators. application to a neuroendocrine system. *SIAM Journal on Applied Dynamical Systems*, 14(2):643–673.

Frank, S. L., Otten, L. J., Galli, G., and Vigliocco, G. (2015). The ERP response to the amount of information conveyed by words in sentences. *Brain and Language*, 140:1 – 11.

Frisch, S., beim Graben, P., and Schlesewsky, M. (2004). Parallelizing grammatical functions: P600 and P345 reflect different cost of reanalysis. *International Journal of Bifurcation and Chaos*, 14(2):531 – 549.

Gayler, R. W. (2006). Vector symbolic architectures are a viable alternative for Jackendoff's challenges. *Behavioral and Brain Sciences*, 29:78 – 79.

Gayler, R. W., Levy, S. D., and Bod, R. (2010). Explanatory aspirations and the scandal of cognitive neuroscience. In Samsonovich, A. V., Johannsdottir, K. R., Chella, A., and Goertzel, B., editors, *Proceedings of the 2010 conference on Biologically Inspired Cognitive Architectures 2010: Proceedings of the First Annual Meeting of the BICA Society*, pages 42 – 51, Amsterdam. IOS Press.

Gigley, H. M. (1985). Computational neurolinguistics: What is it all about? In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, volume 1 of *IJCAI'85*, pages 260 – 266, San Francisco (CA).

Girardi-Schappo, M., Tragtenberg, M., and Kinouchi, O. (2013). A brief history of excitable map-based neurons and neural networks. *Journal of neuroscience methods*, 220(2):116–130.

Gödel, K. (1931). Über formal unentscheidbare Sätze der *Principia mathematica* und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173 – 198.

Golubitsky, M., Stewart, I., Buono, P.-L., and Collins, J. J. (1998). A modular network for legged locomotion. *Physica D*, 115(1-2):56 – 72.

Golubitsky, M., Stewart, I., Buono, P.-L., and Collins, J. J. (1999). Symmetry in locomotor central pattern generators and animal gaits. *Nature*, 401(6754):693 – 695.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *preprint*. arXiv:1511.01427 [cs.NE].

Grefenstette, E., Hermann, K. M., Suleyman, M., and Blunsom, P. (2015). Learning to transduce with unbounded memory. In *Advances in Neural Information Processing Systems*, pages 1819–1827.

Grillner, S. and Zangger, P. (1975). How detailed is the central pattern generation for locomotion? *Brain Research*, 88(2):367 – 371.

Hebb, D. O. (1949). *The Organization of Behavior*. Wiley, New York (NY). Partly reprinted in J. A. Anderson and E. Rosenfeld (1988), pp. 45ff.

Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*. Lecture Notes of the Santa Fe Institute Studies in the Science of Complexity. Perseus Books, Cambridge (MA).

Hinaut, X. and Dominey, P. F. (2013). Real-time parallel processing of grammatical structure in the fronto-striatal system: A recurrent network simulation study using reservoir computing. *PLoS ONE*, 8(2):e52946.

Hinaut, X., Petit, M., Pointeau, G., and Dominey, P. F. (2014). Exploring the acquisition and production of grammatical constructions through human-robot interaction with echo state networks. *Frontiers in Neurorobotics*, 8(16).

Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison–Wesley, Menlo Park, California.

Huyck, C. R. (2009). A psycholinguistic model of natural language parsing implemented in simulated neurons. *Cognitive Neurodynamics*, 3(4):317 – 330.

Ibarz, B., Casado, J. M., and Sanjuán, M. A. (2011). Map-based models in neuronal dynamics. *Physics Reports*, 501(1):1–74.

Ijspeert, A. J. (2008). Central pattern generators for locomotion control in animals and robots: a review. *Neural Networks*, 21(4):642–653.

Jaeger, H. (2001). The echo state approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148:34.

Jansen, B. H. and Rit, V. G. (1995). Electroencephalogram and visual evoked potential generation in a mathematical model of coupled cortical columns. *Biological Cybernetics*, 73:357 – 366.

Joulin, A. and Mikolov, T. (2015). Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in Neural Information Processing Systems*, pages 190–198.

Kleene, S. (1956). Neural nets and automata. *Automata Studies*, pages 3–43.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69.

Kohonen, T. and Somervuo, P. (1998). Self-organizing maps of symbol strings. *Neurocomputing*, 21(1):19–30.

Krupa, M. (1997). Robust heteroclinic cycles. *Journal of Nonlinear Science*, 7(2):129–176.

Lawrence, S., Giles, C. L., and Fong, S. (2000). Natural language grammatical inference with recurrent neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 12(1):126 – 140.

Lewis, R. L. (1998). Reanalysis and limited repair parsing: Leaping off the garden path. In Fodor, J. D. and Ferreira, F., editors, *Reanalysis in Sentence Processing*, pages 247 – 285. Kluwer, Dordrecht.

Li, D. (2014). A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing*, 3.

Lind, D. and Marcus, B. (1995). *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, Cambridge (UK). Reprint 1999.

Lopes da Silva, F. H., Hoecks, A., Smits, H., and Zetterberg, L. H. (1974). Model of brain rhythmic activity: The Alpha-rhythm of the thalamus. *Kybernetik*, 15:27 – 37.

Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531 – 2560.

McClelland, J. L. and Elman, J. L. (1986). The TRACE model of speech perception. *Cognitive Psychology*, 18(1):1 – 86.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115 – 133.

McGhee, R. B. (1968). Some finite state aspects of legged locomotion. *Mathematical Biosciences*, 2(1-2):67–84.

Minsky, M. (1962). Size and structure of universal Turing machines using tag systems. In *Recursive Function Theory: Proceedings, Symposium in Pure Mathematics*, volume 5, pages 229–238.

Minsky, M. L. (1967). *Computation: finite and infinite machines*. Prentice-Hall, Inc.

Mizraji, E. (1989). Context-dependent associations in linear distributed memories. *Bulletin of Mathematical Biology*, 51(2):195 – 205.

Moore, C. (1990). Unpredictability and undecidability in dynamical systems. *Physical Review Letters*, 64(20):2354 – 2357.

Moore, C. (1991). Generalized shifts: unpredictability and undecidability in dynamical systems. *Nonlinearity*, 4:199 – 230.

Neary, T. and Woods, D. (2009). Four small universal Turing machines. *Fundamenta Informaticae*, 91(1):123–144.

Osterhout, L., Holcomb, P. J., and Swinney, D. A. (1994). Brain potentials elicited by garden-path sentences: Evidence of the application of verb information during parsing. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 20(4):786 – 803.

Rabinovich, M. I., Huerta, R., Varona, P., and Afraimovich, V. S. (2008). Transient cognitive dynamics, metastability, and decision making. *PLoS Computational Biology*, 4(5):e1000072.

Schöner, G., Jiang, W. Y., and Kelso, J. A. S. (1990). A synergetic theory of quadrupedal gaits and gait transitions. *Journal of Theoretical Biology*, 142(3):359 – 391.

Sejnowski, T. J. and Rosenberg, C. R. (1987). Parallel networks that learn to pronounce English text. *Complex Systems*, 1:145 – 168.

Shik, M. L., Severin, F. V., and Orlovsky, G. N. (1966). Control of walking and running by means of electrical stimulation of mid-brain. *BIOPHYSICS-USSR*, 11(4):756.

Siegelmann, H. T. and Sontag, E. D. (1991). Turing computability with neural nets. *Appl. Math. Lett*, 4(6):77–80.

Siegelmann, H. T. and Sontag, E. D. (1995). On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150.

Sipser, M. (2006). *Introduction to the Theory of Computation*. Thomson Course Technology Boston.

Smith, J. C., Abdala, A., Koizumi, H., Rybak, I. A., and Paton, J. F. (2007). Spatial and functional architecture of the mammalian brain stem respiratory network: a hierarchy of three oscillatory mechanisms. *Journal of neurophysiology*, 98(6):3370–3387.

Smith, J. C., Abdala, A. P., Borgmann, A., Rybak, I. A., and Paton, J. F. (2013). Brainstem respiratory networks: building blocks and microcircuits. *Trends in neurosciences*, 36(3):152–162.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In Rumelhart, D. E., McClelland, J. L., and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume I, chapter 6, pages 194 – 281. MIT Press, Cambridge (MA).

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46(1-2):159 – 216.

Smolensky, P. and Legendre, G. (2006a). *The Harmonic Mind. From Neural Computation to Optimality-Theoretic Grammar*, volume 1: Cognitive Architecture. MIT Press, Cambridge (MA).

Smolensky, P. and Legendre, G. (2006b). *The Harmonic Mind. From Neural Computation to Optimality-Theoretic Grammar*, volume 2: Linguistic and Philosophic Implications. MIT Press, Cambridge (MA).

Spröwitz, A., Moeckel, R., Vespignani, M., Bonardi, S., and Ijspeert, A. J. (2014). Roombots: A hardware perspective on 3d self-reconfiguration and locomotion with a homogeneous modular robot. *Robotics and Autonomous Systems*, 62(7):1016–1033.

Steil, J. J. (2004). Backpropagation-decorrelation: online recurrent learning with O(N) complexity. In *Proceedings of the 2004 IEEE International Joint Conference on Neural Networks*, volume 2, pages 843–848. IEEE.

Stewart, T. C., Choo, X., and Eliasmith, C. (2014). Sentence processing in spiking neurons: A biologically plausible left-corner parser. In *Proceedings of the Cognitive Science Conference*.

Sukhbaatar, S., Weston, J., Fergus, R., et al. (2015). End-to-end memory networks. In *Advances in Neural Information Processing Systems*, pages 2431–2439.

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

Tabor, W. (2003). Learning Exponential State-Growth Languages by Hill Climbing. *IEEE Transactions on Neural Networks*, 14(2): 444–446.

Tabor, W. (2011). Recursion and Recursion-Like Structure in Ensembles of Neural Elements. *Proceedings of the VIII International Conference on Complex Systems*, 1494–1508.

Tabor, W., Cho, P. W., and Szkudlarek, E. (2013). Fractal analyis illuminates the form of connectionist structural gradualness. *Topics in Cognitive Science*, 5:634 – 667.

Tabor, W., Juliano, C., and Tanenhaus, M. K. (1997). Parsing in a dynamical system: An attractor-based account of the interaction of lexical and structural constraints in sentence processing. *Language and Cognitive Processes*, 12(2/3):211 – 271.

Tsuda, I. (2001). Toward an interpretation of dynamic neural activity in terms of chaotic dynamical systems. *Behavioral and Brain Sciences*, 24:793 – 810.

Turing, A. M. (1937). On computable numbers, with an application to the *Entscheidungsproblem*. *Proceedings of the London Mathematical Society*, 42.

Wegner, P. (1998). Interactive foundations of computing. *Theoretical Computer Science*, 192:315 – 351.

Weir, D. J. (1994). Linear iterated pushdowns. *Computational Intelligence*, 10(4):431 – 439.

Wennekers, T. and Palm, G. (2009). Syntactic sequencing in Hebbian cell assemblies. *Cognitive Neurodynamics*, 3(4):429 – 441.

Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550 – 1560.

Weston, J., Chopra, S., and Bordes, A. (2014). Memory networks. *preprint*. arXiv:1410.3916 [cs:AI].