Spring 6-2-2015

Novel Methods for Learning and Adaptation in Chemical Reaction Networks

Peter Banda
Portland State University

Recommended Citation

Banda, Peter, "Novel Methods for Learning and Adaptation in Chemical Reaction Networks" (2015). *Dissertations and Theses.* Paper 2329.

10.15760/etd.2326

Novel Methods for Learning and Adaptation in Chemical Reaction Networks

by

Peter Banda

A dissertation submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
in
Computer Science

Dissertation Committee:
Christof Teuscher, Chair
Melanie Mitchell
Bart Massey
Niles Lehman
Michael Bartlett

Portland State University
2015

ABSTRACT

State-of-the-art biochemical systems for medical applications and chemical computing are application-specific and cannot be re-programmed or trained once fabricated. The implementation of adaptive biochemical systems that would offer flexibility through programmability and autonomous adaptation faces major challenges because of the large number of required chemical species as well as the timing-sensitive feedback loops required for learning. Currently, biochemistry lacks a systems vision on how the user-level programming interface and abstraction with a subsequent translation to chemistry should look like. By developing adaptation in chemistry, we could replace multiple hard-wired systems with a single programmable template that can be (re)trained to match a desired input-output profile benefiting smart drug delivery, pattern recognition, and chemical computing.

I aimed to address these challenges by proposing several approaches to learning and adaptation in *Chemical Reaction Networks* (CRNs), a type of simulated chemistry, where species are unstructured, i.e., they are identified by symbols rather than molecular structure, and their dynamics or concentration evolution are driven by reactions and reaction rates that follow mass-action and Michaelis-Menten kinetics.

Several CRN and experimental DNA-based models of neural networks exist. However, these models successfully implement only the forward-pass, i.e., the input-weight integration part of a perceptron model. Learning is delegated to a non-chemical system that

computes the weights before converting them to molecular concentrations. Autonomous learning, i.e., learning implemented fully inside chemistry has been absent from both theoretical and experimental research.

The research in this thesis offers the first constructive evidence that learning in CRNs is, in fact, possible. I have introduced the original concept of a chemical binary perceptron that can learn all 14 linearly-separable logic functions and is robust to the perturbation of rate constants. That shows learning is universal and substrate-free. To simplify the model I later proposed and applied the "asymmetric" chemical arithmetic providing a compact solution for representing negative numbers in chemistry.

To tackle more difficult tasks and to serve more complicated biochemical applications, I introduced several key modular building blocks, each addressing certain aspects of chemical information processing and learning. These parts organically combined into gradually more complex systems. First, instead of simple static Boolean functions, I tackled analog time-series learning and signal processing by modeling an analog chemical perceptron. To store past input concentrations as a sliding window I implemented a chemical delay line, which feeds the values to the underlying chemical perceptron. That allows the system to learn, e.g., the linear moving-average and to some degree predict a highly nonlinear NARMA benchmark series.

Another important contribution to the area of chemical learning, which I have helped to shape, is the composability of perceptrons into larger multi-compartment networks. Each compartment hosts a single chemical perceptron and compartments communicate with each other through a channel-mediated exchange of molecular species. Besides the feedforward pass, I implemented the chemical error backpropagation analogous to that of feedforward neural networks. Also, after applying mass-action kinetics for the catalytic reactions, I succeeded to systematically analyze the ODEs of my models and derive the

closed exact and approximative formulas for both the input-weight integration and the weight update with a learning rate annealing. I proved mathematically that the formulas of certain chemical perceptrons equal the formal linear and sigmoid neurons, essentially bridging neural networks and adaptive CRNs.

For all my models the basic methodology was to first design species and reactions, and then set the rate constants either "empirically" by hand, automatically by a standard genetic algorithm (GA), or analytically if possible. I performed all simulations in my COEL framework, which is the first cloud-based chemistry modeling tool, accessible at coel-sim.org.

I minimized the amount of required molecular species and reactions to make wet chemical implementation possible. I applied an automatized mapping technique, Soloveichik's CRN-to-DNA-strand-displacement transformation, to the chemical linear perceptron and the manual signalling delay line and obtained their full DNA-strand specified implementations. As an alternative DNA-based substrate, I mapped these two models also to deoxyribozyme-mediated cleavage reactions reducing the size of the displacement variant to a third. Both DNA-based incarnations could directly serve as blue-prints for wet biochemicals.

Besides an actual synthesis of my models and conducting an experiment in a biochemical laboratory, the most promising future work is to employ so-called reservoir computing (RC), which is a novel machine learning method based on recurrent neural networks. The RC approach is relevant because for time-series prediction it is clearly superior to classical recurrent networks. It can also be implemented in various ways, such as electrical circuits, physical systems, such as a colony of *Escherichia Coli*, and water. RC's loose structural assumptions therefore suggest that it could be expressed in a chemical form as well. This could further enhance the expressivity and capabilities of

chemically-embedded learning.

My chemical learning systems may have applications in the area of medical diagnosis and smart medication, e.g., concentration signal processing and monitoring, and the detection of harmful species, such as chemicals produced by cancer cells in a host (cancer miRNAs) or the detection of a severe event, defined as a linear or nonlinear temporal concentration pattern. My approach could replace hard-coded solutions and would allow to specify, train, and reuse chemical systems without redesigning them. With time-series integration, biochemical computers could keep a record of changing biological systems and act as diagnostic aids and tools in preventative and highly personalized medicine.

DEDICATION

To my father—a biochemist, my first teacher, and the source of my scientific and moral inspiration.

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor Christof Teuscher for his continuous support, guidance, and encouragement. Step by step Christof led me forward and pushed me to fulfill my potential. He taught me the virtues of precision and diligence. He showed tremendous understanding to my ever-changing situation. Indeed it has been a wild ride. He greatly influenced my decision to pursue an academic career.

Many indispensable discussions with Drew Blount have helped to shape this work. This collaboration was fruitful and of far-reaching importance for me. Alireza Goudarzi has always expanded my narrow scientific view and asked fundamental, analytic questions a lot of people would consider overly ambitious. In my eyes he is a true scholar. I also express my appreciation to Josh Moles, who helped me to formulate the idea of chemically implemented delay line, and with whom I share passion for (fancy) new technologies and frameworks.

I deeply appreciated being a part of Teuscher Lab, one the most productive collectives at PSU. I would like to thank my colleagues, especially Jens Bürger for his friendship and to be always there to discuss the everyday scientific and personal struggles in a very open fashion.

I wish to extend my thanks to NSF project "Computing with Biomolecules: From Network Motifs to Complex and Adaptive Systems" for generous support, which made my research possible. I would like to acknowledge all the members of the project, especially

TABLE OF CONTENTS

LIST OF TABLES

LIST OF SYMBOLS AND ACRONYMS

Acronym	Description
CRN	Chemical Reaction Network
AC	Artificial Chemistry
WLP	Weight-Loop Perceptron
WRP	Weight-Race Perceptron
ASP	Asymmetric Signal Perceptron
SASP	Standard Asymmetric Signal Perceptron
TASP	Thresholded Asymmetric Signal Perceptron
AASP	Analog Asymmetric Signal Perceptron
DL	Delay Line
MDL	Manual Signalling Delay Line
BDL	Backpropagation Signalling Delay Line
PDL	Parallel-Accessible Delay Line
FCNN	Feedforward Chemical Neural Network
RNMSE	Root Normalized Mean Square Error
SAMP	Symmetric Absolute Mean Percentage Error
NARMA	Nonlinear AutoRegressive Moving Average
NNLP	Neural Network Linear Perceptron
CHLP	Chemical Linear Perceptron
DNA	Deoxyribonucleic Acid

STATUTORY DECLARATION

I hereby formally declare that I am the sole author of this dissertation and I have not used any source or aid apart from the stated ones. This thesis was neither presented in equal nor in similar form to another examining board at any other university. I cited all used references observing actual academic rules.

In Portland, OR, April 7, 2015   Peter Banda

QUOTE

"Living forms are not in being, they are happening, they are the expression of a perpetual stream of matter and energy which passes through the organism and at the same time constitutes it."
Ludwig von Bertalanffy (1901-1972)

1

INTRODUCTION

Our bodies consist of cells, powerful chemical machines, capable of dynamic resource distribution, repair, and reproduction. Biochemical media, especially DNA strands, offer generic combinatorial power to model a wide variety of computational systems. Chemistry supports information processing due to inherent parallelism, massive interactivity, redundancy, and asynchronicity [11, 42, 82]. Biomolecular systems have successfully tackled several computing problems, including the traveling salesman problem [10], 3-SAT [28], maximal clique [119], chess [50], and tic-tac-toe [145]. However, attempts to build a programmable molecular automaton, that is, an automaton with more than one hard-wired purpose failed, or had limited scope and no reusability [23, 41, 121]. The main challenge is that the integration of the forward-pass, where the input species transform to the output, with the backward-pass, where a user's action affects the concentrations of species that define the system's functionality, requires reactions with complicated and timing-sensitive relations that involve positive and negative catalytic feedback loops.

State-of-the-art biochemical systems for medical applications and chemical computing are application-specific and cannot be re-programmed or reused once fabricated. Typically these systems are one-shot-only so after they do certain work they cannot recover their state and must be discarded. After injection *in vitro* or *in vivo*, nano-scale chemical machines [157] are difficult to interface with and control. To address that, new chemical

systems must function not only in idealized well-known lab settings, but also in previously unanticipated environments. Rather than following static rules, their response must be robust and adaptive. Adaptive chemical systems would decide autonomously and learn new environments through reinforcements in response to external stimuli. We could imagine that in the future millions of molecular spiders [137] would help our immune system fight viruses, deliver medications, fix broken cells, etc. For that vision to become true I wish to understand how autonomous learning and adaptation can be incarnated in a chemical medium.

The main goal of this dissertation is to introduce a flexible programmable template for synthetic biochemistry that could find use in a variety of applications, such as drug delivery [92], pattern recognition, smart medication, and chemical computing [42]. Currently the design of chemical systems is time-consuming and costly due to the complexity of molecular interactions, and undesired leakage and crosstalk. Often the only methodology available is a tedious trial-and-error approach. I argue that by understanding and developing adaptation in chemistry, we will be in a position where, instead of multiple systems with hard-wired purpose, we may design a single programmable template that can be trained (and re-trained) to achieve a desired input-output function profile.

Another motivation is to fit learning into the broader definition of life and challenge its conventional meaning. The chemical medium is the basis of everything living on this planet; therefore, it is the most natural choice for implementing an artificial life. Learning allows organisms to generalize and predict the observed environment, and therefore to gain a competitive advantage for their survival and reproduction. As I will show, the feedback-based adaptation requires special molecular species, meta catalysts, that drive the synthesis of underlying catalysts, whose concentrations represent the system's state. I will also place the presented chemical learning models in a wider context to see how their

structure compares to the chemical organization of living systems, especially Rosen's metabolic-repair systems [131].

In complex biological organisms learning is carried out by neurons organized into networks and ultimately brains. Artificial neural network theory [62] investigates such systems by applying the circuit-based abstraction, wherein inputs and so-called synaptic weights are integrated and processed by the activation function to produce the output signal (an action potential). The question I ask is whether a system at the molecular level, an order of magnitude simpler than those of a neuro-biological origin, could adapt as well. Note that chemistry acts on different premises from formal neural circuits, e.g., it is diffusive, lacks topology, and its dynamics preserve matter, so we could not map formal neural models to chemistry in a straightforward one-to-one manner.

1.1 DISSERTATION CONTRIBUTIONS

In the dissertation I explore various approaches to learning and adaptation in the widely used formalism of chemical reaction networks, realistic yet abstract simulated chemistries. My work is the first successful scientific endeavour with constructive proofs to define "chemical learning" as a new interdisciplinary field at the threshold of machine learning, neural networks, chemical reaction networks, biochemistry, and perhaps control theory.

The chemical building blocks introduced in this dissertation fit together to obtain more advanced systems with sufficient computational and representational power to serve different bio-computing applications, such as adaptable and programmable decision-making *in vivo*, pathogen detection, or processing and monitoring concentration time series. As opposed to hard-wired chemical circuits, my systems can learn and therefore morph functionally to many target behaviors defined by an input-output profile.

To show that my chemical reaction networks, whose species are symbolic and unstructured, could be translated to wet chemicals, I implemented two of my models, a chemical perceptron and a chemical delay line, at a low-level by two DNA reaction primitives: DNA strand displacement and deoxyribozymes. These DNA incarnations carry out the essential reaction structure and dynamics of the original chemical systems. Even though my DNA-implemented chemical learner is substantially larger than single-purpose chemical systems, its reusability and flexibility will eventually reduce the design and fabrication cost in the long term.

The detailed contributions of my dissertation are as follow.

- I conceptualized the representation of binary and signed real numbers by means of species concentrations and identified two basic categories—symmetric and asymmetric encodings (Chapter 3).

- I introduced the very first model of a simulated chemical system capable of autonomous learning. I coined the term *binary chemical perceptron* because its two initial variants, weight-loop perceptron and weight-race perceptron, can solve all 14 linearly separable logic functions perfectly as a formal perceptron. Both models are robust to the perturbation of rate constants and employ a symmetric representation, however, they treat the roles of input and weight species inversely. These perceptrons are trained by supervised learning through injections of the input and desired-output pairs [21] (Chapter 4).

- I further simplified the chemical perceptron and reduced its size by 50% by employing asymmetric chemical arithmetic with a novel representation of negative numbers and subtraction and introduced a binary asymmetric signal perceptron. As opposed to previous models, the asymmetric signal perceptron learns by penalty

signal, which fits better its asymmetric design. As before, the perceptron can learn all 14 linearly separable functions [22] (Chapter 4).

- I incorporated the minimal bistable chemical system with 4 reactions (formulated by Wilhelm [159]) to the asymmetric signal perceptron design to obtain an active thresholding of the output, such that its final concentration is effectively binary, i.e., low (zero) or high (one) [22] (Chapter 4).

- I modeled an analog chemical perceptron called *analog asymmetric signal perceptron* by adjusting a design of the binary asymmetric signal perceptron (e.g., adding input-weight contribution species and replacing a penalty signal with desired output) and showed that it can learn various linear and nonlinear functions of two inputs with an error (RNMSE) in the range $(0.103, 0.0.378)$ [20] (Chapter 5).

- To show chemistry is capable of explicit memorization, I implemented—in collaboration with Josh Moles—two sequential delay lines as chemical reaction networks. Chemical delay lines store the past inputs over a sliding window of a given size and feed them to an underlying system [114] (Chapter 6).

- I addressed the issues of previous chemical delay lines, in particular, their sequentiality, latency, and large number of operating copy signals, in the design of a new parallel-accessible delay line. It functions optimally, i.e., aside from a constant (arbitrary short) waiting time, mimics an algorithmic delay line perfectly [19] (Chapter 6).

- I demonstrated the modularity and reliability of the parallel-accessible delay line by an integration with an analog asymmetric signal perceptron of two to five inputs. The integrated memory-extended chemical analog perceptron tackled four

time series: linear moving average, moving maximum, and benchmark NARMA2 and NARMA10 tasks [19] (Chapter 6).

- To demonstrate composability of chemical perceptrons, I—in collaboration with Drew Blount—created the concept of a feedforward chemical neural network [27], which consists of hierarchical compartments communicating with each other through channel- mediated exchange of chemical species. A significant contribution is the implementation of error backpropagation analogous to that of formal feedforward neural networks (Chapter 7).

- By combining three analog asymmetric signal perceptrons and adjusting their error production and propagation—in collaboration with Drew Blount—I created a feedforward chemical network where a skin compartment with an outer neuron contains two subcompartments (two hidden neurons). This basic topology analogical to the 2-2-1 formal feedforward neural network allowed the system to learn all binary functions with 99.88% average success rate, including XOR and XNOR. Note that because XOR and XNOR are nonlinearly separable, they are beyond the capabilities of a single binary chemical perceptron. The same property holds for formal neural network perceptrons [27] (Chapter 7).

- To complement purely statistical simulations and provide an insight into chemical learning, I rigorously analyzed the differential equations of analog asymmetric signal and chemical linear perceptrons and derived closed or approximative formulas for the input-weight integration and weight update. I showed the formulas of the linear chemical perceptron match those of the formal neural network linear perceptron aside from bounding the weights and a normalization of the inputs during a weight update. These analytical findings established a solid connection between

adaptive chemical reaction networks and neural networks (Chapter 8).

- I introduced an annealed weight update using an error decay species, which amplifies or reduces the error obtained from the output and the desired output. Annealed weight updating benefits the weight convergence and overall performance (Chapter 8).

- I compared the performance of the chemical linear perceptron, the neural network linear perceptron, and the analog asymmetric signal perceptron on the previously used 6 static functions of two inputs and 4 time series with an algorithmic delay line of size 2 to 20. I showed that the performance of the chemical linear perceptron and the neural network linear perceptron equal and exceeds the analog asymmetric signal perceptron by 94 or 437 times on average for the static functions depending on the error metrics. For the time series, the error is reduced by 8.37 (or 15.24) times. (Chapter 8).

- To demonstrate the feasibility of wet chemical applications, especially in the areas of smart drug delivery and chemical computing, I provided biochemical specification of the linear chemical perceptron and the manual signalling delay line by using DNA strand displacement and deoxyribozyme reaction primitives. These could directly serve as blue-prints for synthetic realizations of my models (Chapter 9).

- I implemented the first web-based chemistry simulation framework, COEL, available at coel-sim.org. Its most prominent features include ODE-based simulations of chemical reaction networks and multicompartment reaction networks, with rich options for user interactions, optimization of rate constants based on genetic algorithms, expression validation, an application-wide plotting engine, and SBML/

Octave/Matlab export. I programmed COEL in Scala and Java, and employed primarily Grails, Spring, Hibernate, and GridGain technology stack. A visually pleasing and intuitive user interface, simulations that run on a large computational grid, reliable database storage, and transactional services make COEL ideal for collaborative research and education. COEL has currently around 40 users and is provided openly for educational and research purposes. Note that the efforts to make the project open source are under way [17] (Chapter 10).

- I implemented a parser of domain-specified DNA strands represented as strings in Visual DSD syntax [91]. The parser outputs SVG visualizations of DNA strands which are embedded but could be exported [17] (Chapter 10).

- I programmed the well-known Soloveichik's transformation that automatically compiles any chemical reaction network to DNA strand displacement circuit with the species specified by DNA domains. To the best of my knowledge, COEL is the only CRN simulation framework to provide that. By an integration with my DNA strand parser the output of the compilation process is a list of visualized DNA strand displacement reactions and species with a convenient export functionality. This feature could be of far reaching importance for general scientific community [17] (Chapter 10).

- In this field I published 4 journal papers [21, 22, 27, 114] (including one under review), 3 conference papers [17, 19, 20], and one extended abstract [18]. I held a poster presentation at 4 venues and had 3 oral conference presentations.

2

BACKGROUND AND RELATED WORK

Learning and adaptation, along with homeostasis and growth, are among the fundamental characteristics that life in the most general sense exhibits [25, 88]. They represent the ability of an individual to alter its response and decision-making by using feedback from the environment. This ability lets individuals adjust and escape predefined behavioural patterns given by evolution, i.e., adaptation at the level of the population. Learning has been a vibrant topic in the artificial life and neural network communities for over two decades. It has been realized by means of neural networks [62, 130], various forms of evolutionary algorithms [112, 113], and reinforcement learning [147], in which agents learn from the consequences of their actions through rewards. Some applications of learning include path finding problems [80], multi-agent systems [101], and robotics [31].

2.1 NEURAL NETWORK THEORY

Artificial neural networks [62] are inspired by the coarse-grained behavior of biological neurons in the brain and a desire to emulate and understand its computational power. Neural network theory formalizes the functioning of biological neurons as linear-integration circuits with a threshold, sigmoid, or other monotone activation function. Note that this overview section does not cover all neural network models thoroughly, but describes only

those that are a primary inspiration for the implementation of so-called chemical percep-
tron and mutlicompartment chemical network presented in Chapters 4, 5, and 7 respec-
tively.

2.1.1 Perceptron

The perceptron, introduced by Rosenblatt [132], is an early type of artificial neural net-
work [130]. Despite all simplifications, the perceptron is capable of non-trivial learning
and forms the basis of more complex feed-forward neural networks (Section 2.1.2).

A perceptron processes a vector of input signals $\mathbf{x} = (x_1, \ldots, x_n)$, $x_i \in \mathbb{R}$, and produces
one output y based on the setting of its weights $\mathbf{w} = (w_0, w_1, \ldots, w_n)$ as shown in Figure
2.1. More precisely, a perceptron first calculates the linear integration (the dot product)
of weights $\mathbf{w}$ and inputs $\mathbf{x}$ as $v = \sum_{i=0}^{n} w_i \cdot x_i$, and then passes the result v to an activation
function $\varphi : \mathbb{R} \rightarrow [0, 1]$ or $\varphi : \mathbb{R} \rightarrow [-1, 1]$, which produces the final output y. The weight
w_0, called bias or offset, always contributes to an output, since its associated input x_0 is
constant 1.
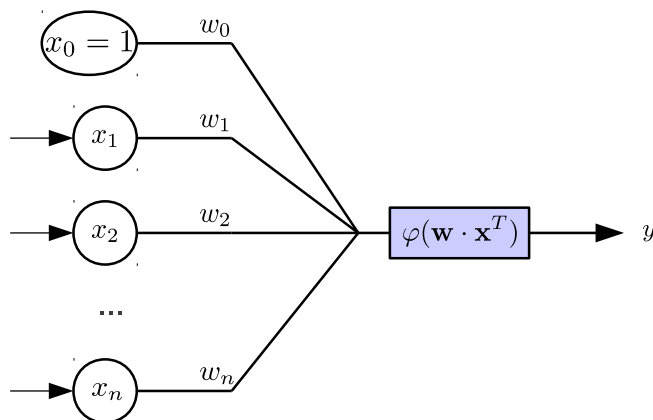
Figure 2.1: Model of a perceptron. An activation function φ processes the dot product of weights and inputs
$\mathbf{w} \cdot \mathbf{x}^T$, producing output y.

A perceptron can classify only linearly separable functions [111]—functions in which a straight line, or in the general case, a hyperplane can divide the inputs into two classes. By combining several perceptrons, we can construct a multilayer perceptron network, also known as a multilayer feed-forward network [62] that overcomes the linear separability problem and in fact becomes a universal approximator [72].

Supervised Learning

Perceptron learning [130] is a type of supervised Hebbian learning [63] where a training data set $T = \{(\mathbf{x_1}, d_1), \ldots, (\mathbf{x_m}, d_m)\}$, consisting of input-output pairs, characterizes the target behavior of the system. During each step of the learning process, a perceptron absorbs one training sample $(\mathbf{x}, d)$. If in the current state of the perceptron (i.e., its current weights) there is a discrepancy between its actual output y and the desired output d, the error is fed back to the perceptron and triggers an adaptation of the participating weights. The adaptation of a weight w_i for the training sample $(\mathbf{x}, d)$ at time t is defined as $w_i(t+1) = w_i(t) + \alpha(d - y(t))x_i$, where $\alpha \in (0, 1]$ is the learning rate. If an error is detected, that is, if $|d - y| > 0$, the weight w_i shifts toward the desired output proportionally to its input signal x_i. If an input $x_i = 0$, the weight w_i is not involved in the global output y and therefore stays unaltered. Initially, the weights are set to small random values. The process of weight adaptation continues until the cumulative error of several consecutive training samples drops below the error threshold, or alternatively a fixed number of iterations is reached. After a training, performance (accuracy) is evaluated against a test set, which might include different input-output pairs from the target domain.

Learning By Reinforcement

An alternative learning method to supervised learning, that is more biologically plausible [106] and widely used for agent-based modeling, is reinforcement learning [147]. In reinforcement learning agents learn their expected behaviour from the consequences of their actions through rewards (positive reinforcements) and/or penalties (negative reinforcements). To replace the classical perceptron learning algorithm with reinforcement specified as a single penalty signal, the adaptation of weight w_i for the penalized perceptron is $w_i(t+1) = w_i(t) \pm \alpha\, b\, x_i$, where b is the constant penalty signal. The perceptron itself must determine whether the weight should be increased or decreased as a consequence of penalization, i.e., the production of incorrect output.

Table 2.1: The relation of an input pair x_1 and x_2 to output y in the two-input binary perceptron, where w_0, w_1 and w_2 are weights, and θ is a threshold.

x_1	x_2	y
0	0	$w_0 > \theta$
1	0	$w_0 + w_1 > \theta$
0	1	$w_0 + w_2 > \theta$
1	1	$w_0 + w_1 + w_2 > \theta$

Two-Input Binary Perceptron

The two-input binary perceptron with a threshold activation function outputs one if the inner product $w_0 + w_1 x_1 + x_2 w_2$ is greater than the threshold θ, zero otherwise. Because the input is binary, the linear integration collapses to the four cases summarized in Table 2.1. The two-input binary perceptron can learn all 14 linearly separable binary functions, i.e., all two-input binary functions except `XOR` and `XNOR`.

Now, we can ask the question what happens if we restrict some weights only to positive and some weights only to negative values?0 For instance, let us assume w_0 is negative

Table 2.2: Overview of the modeling capabilities of the two-input binary perceptron restricted to positive or negative values of weights w_0, w_1, w_2.

Weights			Binary Functions															
w_0	w_1	w_2	FALSE	NOR	NCIMPL	NOT X1	NIMPL	NOT X2	XOR	NAND	AND	XNOR	PROJ X2	IMPL	PROJ X1	CIMPL	OR	TRUE
⊖	⊖	⊖	×															
⊖	⊖	⊕	×				×								×			
⊖	⊕	⊖	×		×								×					
⊖	⊕	⊕	×								×		×		×		×	
⊕	⊖	⊖		×		×		×		×								×
⊕	⊖	⊕						×								×		×
⊕	⊕	⊖				×								×				×
⊕	⊕	⊕																×

and w_1 and w_2 are positive. Then the weights $w_0 = -10, w_1 = 7, w_2 = 9$ model the AND function and the weights $w_0 = -10, w_1 = 12, w_2 = 13$ the OR function. However, no combination of negative w_0 and positive w_1 and w_2 weight values can represent the NAND function. Table 2.2 summarizes the limitations of all sign-weight combinations for a representation of logic functions. It shows that each weight must support both positive and negative values to implement a perceptron that can encompass all 14 linearly separable binary functions. This is especially important for an implementation of the chemical perceptron (Chapter 4 and 5), because it implies that the representation of negative numbers could not be avoided.

2.1.2 Feed-Forward Neural Network

A feed-forward neural network [62] is an acyclic neural network consisting of perceptrons (neurons) organized into layers. For the so-called *forward pass*, an input x is fed into the input layer neurons, whose outputs become inputs for the next (hidden) layer, and so on until the final (output) layer containing a single output neuron produces the output y. The

formula for a neuron i is

$$\varphi(\sum_j x_j w_{ij}), \qquad \boxed{2.1}$$

where w_{ij} is the weight coming from neuron j (from the previous layer) to the neuron i, φ is an activation function, and x_j is the output of neuron j or the input if it is located in the input layer. This is a direct extension of the simple perceptron input-weight integration presented in Section 2.1.1.

The weight adaptation or *backward pass*, however, gets more complicated than in a simple perceptron case (Section 2.1.1). The so-called delta rule consists of two stages. The first stage calculates the cumulative error of each neuron through backpropagation. Starting with the output perceptron, the error δ of is calculated as $d - y$, where d is the desired output. Then, working backwards layer by layer we calculate each remaining neuron's error. Now consider the single perceptron i in the hidden layer. The error that propagates from the neuron k in the next layer is weighted. The global error for the perceptron i is then the linear combination of the weighted errors of the next layer's perceptrons connected to the perceptron j and the sensitivity (first derivative) of the activation function φ with respect to the input-weight dot-product v:

$$\delta_i = \varphi'(v_i) \sum_i \delta_k w_{ki}, \qquad \boxed{2.2}$$

where the derivation of the activation function $\varphi'(v_i)$ with $v_i = \sum_j x_j w_{ij}$ is $y(1 - y)$ for a logistic function $\varphi(v) = (1 + e^{-v})^{-1}$ and $(1 + y)(1 - y)$ for a hyperbolic tangent function $\varphi(v) = \tanh(v)$.

Once all of the errors have been calculated, each perceptron's weights are updated as

$$\Delta w_{ij} = \alpha \delta_i x_j. \qquad (2.3)$$

2.2 INTRODUCTION TO GENETIC ALGORITHMS

Genetic algorithms (GA) [112] were originally introduced by Holland [68] as a stochastic optimization tool inspired by the Darwinian evolution. GA is an iterative process that intelligently searches through a space of possible solutions. GAs are popular and widely applied for many scientific or technological problems [12, 14, 16, 58].

A GA operates on a population of chromosomes, which encode possible solutions for a given problem and are represented by vectors. The initial population is usually generated at random or using a heuristic. During each generation (evolutionary step), the GA calculates the fitness of each chromosome, which reflects how well the chromosome solves a given problem. For instance, the fitness could be a fraction of the correctly classified instances to the number of trials.

The best chromosomes act as parents of new individuals in the next generation. Chromosomes can be selected to reproduce standardly either by elite or roulette method [112]. The elite method selects deterministically a certain number of the fittest chromosomes. The roulette method selects chromosome with a probability proportional to their fitness. In the roulette method, the selection probability could be adjusted by a fitness renormalization.

Reproduction can be sexual or asexual. In the former case, crossover between two selected parent chromosomes can be either one-point (i.e., in chromosomes of length n, the child's first $p \leq n$ genes are from one parent and the last $n - p$ are from the other),

or a probabilistic shuffle. Also, crossover could be conditional, hence it occurs with a probability $p_{cross} \leq 1$, otherwise new off-springs are exact copies of their parents. The mutation operation alters certain bits in newly created chromosomes. The bit alternation can be produced either by a full replacement with a newly generated bit, or for Integer or Real numbers a new bit can be generated by a perturbation. The number of bits the mutation changes depends on the mutation type: one-bit, two-bit, exchange and per-bit. Similarly to crossover the mutation could be conditional.

As presented in Figure 2.2 an evolutionary cycle consists of fitness calculation, selection, crossover and mutation, and it repeats until the stop-criterion, such as, the target (maximum) fitness, or alternatively a fixed (maximal) number of generations is reached. For more information about GA and evolutionary dynamics we advise the reader to refer to the many excellent textbooks that already cover these areas, e.g., [40, 112].
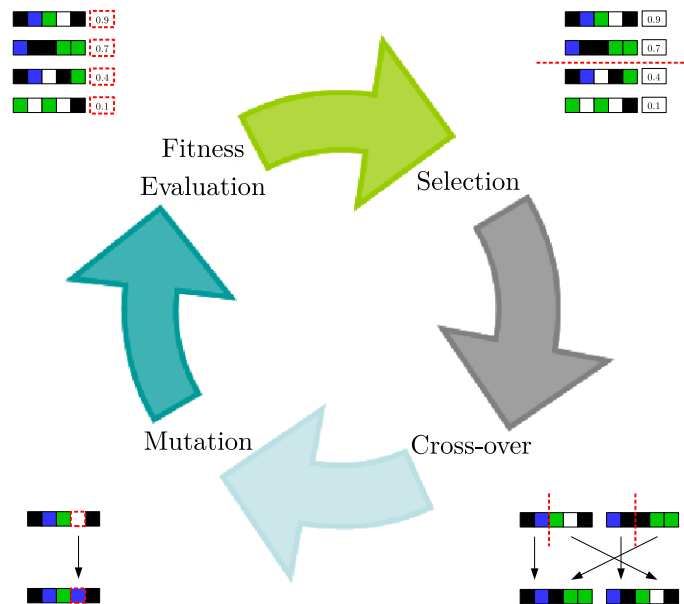
Figure 2.2: A genetic algorithm as a cyclic process.

2.3 CHEMISTRY

Chemistry is the study of matter and its transformations. Depending on the desired precision the focus of chemistry could vary from the small to the large matter. The small matter, such as protons and electrons, is accompanied with valence relations and further down with the equation of the wave motion. The larger matter, such as atoms and molecules, follows mass-action law, which describe the concentration change over time. Even larger complex biochemical molecules, in particular DNA, could form complicated 3-D structures, such as origami [105]. Throughout centuries, researchers collected chemistry knowledge based on empirical study: experiment, observation, and generalization. Chemistry describes and also helps us to make predictions about the reactions not conducted before. Throughout the reminder of this dissertation we make a distinction between chemistry used as a set of mathematical laws and wet chemistry, which refers to actual experiment in a lab and embraces what *really* happens physically.

Chemistry neighbors and bridges physics on the lower and biology on the upper end of the granularity spectrum. For different applications we need to balance the complexity of laws we want to keep, and the severity of mismatch with the *wet* chemistry we can afford. With the offspring of computers we could propose a set of reactions and molecular species and simulate their dynamics without anchoring our models in physical reality. The goal is to obtain a system that carries out some form of information processing, or sensing. At this level chemistry does not describe, but rather *prescribes* how chemicals should behave, keeping some part of their specification open. That means the proposed abstract chemicals still follow a set of chemical laws, but they do not relate directly to existing molecular species, natural or synthetic. Of course the more abstract the chemical design, the larger discrepancy and the more difficult the translation to a wet chemical implementation, since

the space of wet molecular species to consider for possible substitution of the abstract species expands.

2.3.1 Chemical Reaction Network

In this thesis I employ almost exclusively the formalism of *Chemical Reaction Network* (CRN) [48, 71]. A CRN is the standard framework for representing chemistry that consists of the finite set of molecular species and the finite set of reactions paired with corresponding rate constants [47, 49]. CRN represents an unstructured macroscopic simulated chemistry, hence, the species labeled with symbols are not assigned a molecular structure yet. More importantly, CRN lacks the notion of space—in a well-stirred tank, the probability that a molecule is involved in a reaction does not depend on its position, but solely on its type. Consequently, the state of the system is a vector of species' concentrations. Note that we use concentration as a dimensionless quantity, and a wet chemical implementation could scale the concentrations to micro molar (μM) or nano molar (nM) as needed. The container of chemical species specified by their concentrations, i.e., the whole system, is often referred to as a tank, a reactor, or a solution.

Each reaction is an ordered pair $k_1X_1 + \ldots + k_nX_n \rightarrow l_1Y_1 + \ldots + l_mY_m$, where species X_i are reactants, species Y_i are products, and constants k_i and l_i are called stoichiometric factors. For instance, a reaction $A + B \rightarrow C$ describes a transformation of species A and B that bind together to form species C. Note that a legal reaction could have no reactants or no products. For that purpose we introduce a special no-species symbol λ to represent a formal annihilation $A + B \rightarrow \lambda$ or a decay $A \rightarrow \lambda$. Mass conservation states that matter cannot be destroyed nor created—in a closed system the matter consumed and produced by each reaction is the same. Annihilation and decay as we defined them seem to violate that, however, in the chemical analogy, λ does not signify a disappearance

of matter but simply an inert species, effectively absent from the system of chemical interactions. Similarly we interpret a reaction $\lambda \rightarrow A$ as an influx of A rather than a creation of a molecule A from nothing.

Each reaction has a rate, which defines the strength of the reaction's contribution to the production or consumption of particular species over time. The rate expressions are not arbitrary, but are prescribed by kinetics laws. More precisely, the three most common kinetics are mass-action law [13,49] for ordinary, Michaelis-Menten kinetics [38,108] for catalytic, and linear uncompetitive kinetics for inhibitory reactions.

The mass-action law [13, 49] states that the rate of a reaction is proportional to the product of the concentrations of the reactants. Hence the reaction rate, the *speed* of the reaction application, is assumed to be linearly dependent on the concentration of reactants (Figure 2.3(a)). For an irreversible generic reaction $aS_1 + bS_2 \rightarrow P$, the rate is given by

$$r = \frac{d[P]}{dt} = -\frac{1}{a}\frac{d[S_1]}{dt} = -\frac{1}{b}\frac{d[S_2]}{dt} = k[S_1]^a[S_2]^b, \qquad \boxed{2.4}$$

where $k \in \mathbb{R}^+$ is a reaction rate constant, a, b are stoichiometric constants, and $[S_1]$, $[S_2]$ are a concentration of reactants (substrates) S_1, S_2 respectively. Note that in wet chemistry for some cases the coefficients in the rate law might not equal the stoichiometric constants. We will however assume that our reactions are all "well-behaved".

Besides being a reactant or a product, a species can take on two other roles in a reaction. A *catalyst* is a substance that increases the rate of a reaction without itself being altered. To incorporate catalysts (or enzymes) in reaction rates, we follow Michaelis-Menten enzyme kinetics [38, 108] (Figure 2.3(b) and 2.3(c)). Let $E + S \rightleftharpoons ES \rightarrow E + P$ be a catalytic reaction written compactly as $S \xrightarrow{E} P$, where E is a catalyst, S is a substrate, P is a product, and ES is an intermediate enzyme-substrate binding species. Michaelis-

(a)

(b)

(c)
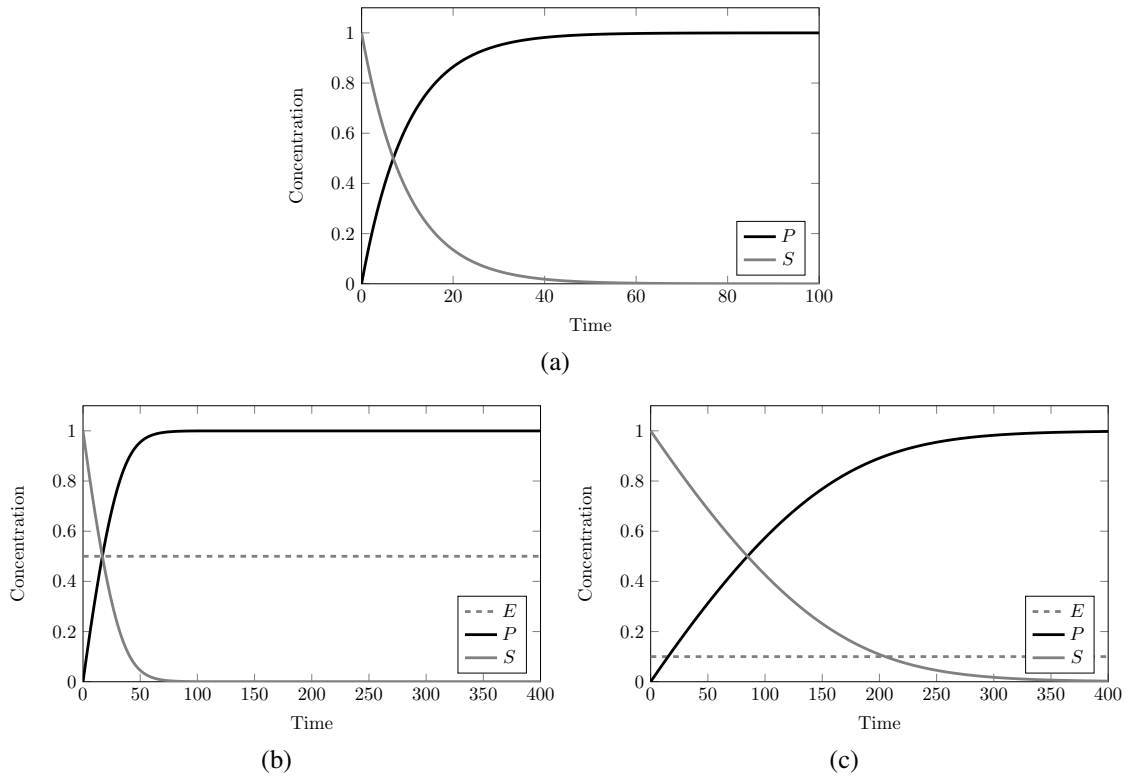
Figure 2.3: Example traces of (a) $S \rightarrow P$ reaction driven by mass-action kinetics and (b-c) catalytic $S \xrightarrow{E} P$ reaction with catalyst E driven by Michaelis-Menten kinetics. The substrate S transforms to the product P faster for a higher concentration of $[E] = 0.5$ (b) as opposed to $[E] = 0.1$ (c). The reaction rate of (a) is $k = 0.1$ and (b-c) $k_{cat} = 0.1$, $K_m = 0.5$.

Menten kinetics assumes that the substrate S is in instantaneous equilibrium with the enzyme-substrate complex ES. This assumption is called the quasi-steady-state approximation, which holds if the first reaction $E + S \rightarrow ES$ is substantially faster than the second one $ES \rightarrow E + P$. Another assumption is that the enzyme concentration $[E]$ is much smaller than the substrate concentration $[S]$. An overall reaction rate for the P production is

$$r = \frac{d[P]}{dt} = -\frac{d[S]}{dt} = \frac{k_{cat}[E][S]}{K_m + [S]}, \qquad (2.5)$$

where $k_{cat}, K_m \in \mathbb{R}^+$ are rate constants. Michaelis-Menten kinetics can be also ex-

panded to the multi-substrate case [94]. An alternative to Michaelis-Menten kinetics, without any assumptions about the enzyme concentration or the speed of the reactions, is to use mass-action kinetics for two partial, associative and disassociative reactions, $E + S \rightleftharpoons$ and $ES \rightarrow E + P$, which yields

$$
\begin{aligned}
\frac{d[P]}{dt} &= -k_f[E][S] + k_r[ES] + k_{cat}[ES] \\
\frac{d[S]}{dt} &= -k_f[E][S] + k_r[ES] \\
\frac{d[ES]}{dt} &= k_f[E][S] - k_r[ES] - k_{cat}[ES] \\
\frac{d[P]}{dt} &= k_{cat}[ES],
\end{aligned}
\tag{2.6}
$$

where k_f is a forward association rate, k_r is a reverse association rate, and k_{cat} is a disassociation rate.

An *inhibitor* is a substance that retards the rate of a reaction without itself being consumed. Several types of inhibition exist, however, our CRN is restricted to the simplest one, known as linear uncompetitive inhibition [94]. The reaction rate of $I + S \rightarrow I + P$, where I is an inhibitor, and k and K_i are rate constants is

$$
r = \frac{d[P]}{dt} = -\frac{d[S]}{dt} = \frac{k[S]}{1 + K_i[I]}.
\tag{2.7}
$$

By applying rate laws over all reactions we obtain the change of a concentration of molecular species described as a system of deterministic *ordinary differential equations* (ODEs). Since it is, in general, impossible to find an analytical solution of such a system explicitly, we employ numerical integration of the ODEs, which delivers an approximative solution [135]. To simulate the learning protocol, we use either 0.5, 0.1, or 0.05-step Runge-Kutta4 (RK4) numerical integration [69,142] of the rate ODEs. The Runge-Kutta4

solver provides a good balance between the result quality and the simulation cost. To assure stability of the RK4 solver, i.e., to bound the derivations of overall concentration functions, we restrict the rate constants and the initial concentrations appropriately.

An alternative to deterministic ODE-driven chemistry is the Gillespie method [56], which simulates each reaction step stochastically on a molecular level [79,152]. Although it is more realistic physically, it is computationally more expensive. As the number of molecules increases, the stochastic results will converge to the deterministic solutions. Hence, if we scale up the number of molecules in our system, the deterministic (numerical) simulations would match stochastic (and therefore also real) chemistry quite well.
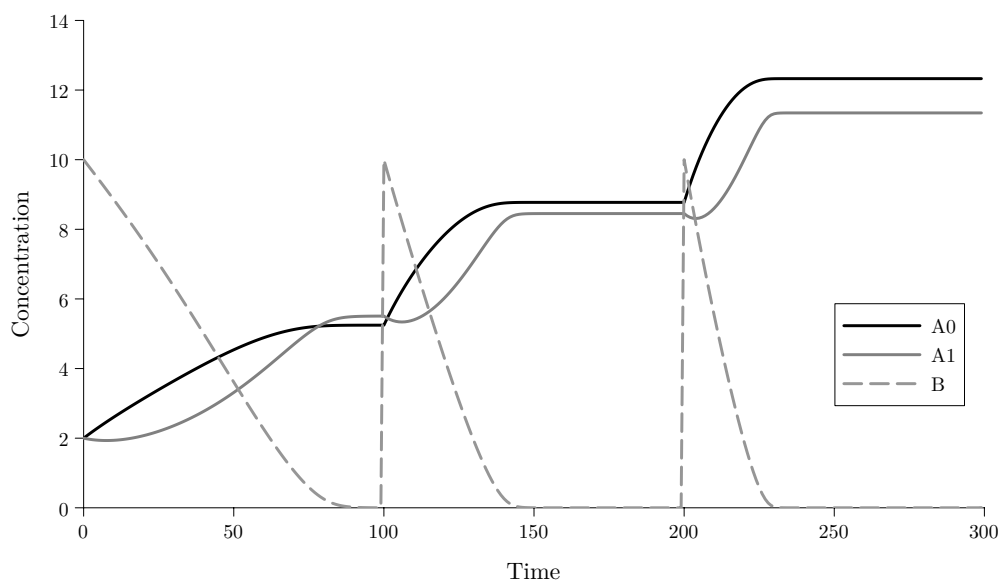
Figure 2.4: A concentration trace of a chemistry with species $S = \{A_0, A_1, B\}$ and reactions $R = \{R_0 : A_1 + B \rightarrow A_0, R_1 : B \xrightarrow{A_0} A_1\}$ using the rate constants $k = 0.00325, k_{cat} = 0.025$, and $k_m = 0.5$. Injections are provided at time step t_0: $[A0] = [A1] = 2$, and $[B] = 10$, t_{100}: $[B] = 10$, and t_{200}: $[B] = 10$. The output interpretation $\max([A1]) > \max([A0])$ produces the output sequence $1, 0, 0$ (from left to right).

2.3.2 Interaction and Interpretation

The state of a chemical system is a vector of species concentrations. To interact with the system we need to perturb its state—the concentration of specific species that usually represents the input. An interaction emulates a step in the execution of an experimental protocol, where at a certain time the person performing the chemical experiment injects or removes substances into or from a tank. It is modeled by instantaneously changing the concentration of a species. Concentrations can be modified multiple times, not just at time t_0. For iterative processes, such as learning, it is useful to define repetitive interactions, in which a sequence of interactions repeat in a loop at predefined time intervals. For the output interpretation we must carry out a reverse translation: we map the concentrations of the designated output species at certain time or time period to the formal variable. The output interpretation usually follows the input injection, and similarly to interactions, interpretations can be repetitive.

For example, let R_0 be a reaction $A_1 + B \rightarrow A_0$, and R_1 be a reaction $B \rightarrow A_1$ catalyzed by A_0 using the species set $S = \{A_0, A_1, B\}$. Figure 2.4 presents a concentration trace of species from S driven by the reactions R_0 and R_1. Interactions occur at time t_0, t_{100}, and t_{200} as described in the caption. By applying the output interpretation defined as $\max([A_1]) > \max([A_0])$ on the intervals $t_0 - t_{99}, t_{100} - t_{199}$, and $t_{200} - t_{299}$, we translate the output to sequence $1, 0, 0$.

2.3.3 Artificial Chemistry, a Historical Overview

Artificial chemistry (AC) as defined by Dittrich [44] spans many models from grammars to more substrate-specific structured or spatial chemistries. At the beginning of 2000, there was a tendency to incorporate general simulated chemistry (CRN) to the artificial

chemistry due to the influence of the artificial life community. Since a lot of systems with very loose relation to chemistry have been called artificial chemistries, it became difficult to systematically define this field. For instance *cellular automata* (CA) [160] and Fontana's AlChemy [54] are often considered artificial chemistries, even though they do not follow any exact chemical or physical laws. They better fit into the field of multi-agent systems or complex networks in the former and random lambda calculus in the later case. Initially the abstract rewriting system on multisets [118, 149] occupied the core part of artificial chemistry, however, later the focus shifted to more grammar-like forms and a large part of the artificial chemistry field turned into P systems [125].

P Systems [120, 125, 126] consider heavily abstracted chemical systems where a different reactions occur in different cells, with communication between cells via membrane permeation. The main difference between CRN and P Systems is that P Systems that represent chemical objects by strings over a defined alphabet, which react through rewriting rules taken from Chomsky grammars. P systems do not model reaction rates, have a more grammar-like update, and manipulate discrete symbols. P Systems successfully demonstrated the computational power of formal grammars of a specific type, but significant omissions, such as mass action kinetics, separate them from chemical computing in practice.

Because of the term overloading and the unclear distinction between rule-based systems and artificial chemistry, there is tendency to drop the adjective *artificial* or substitute it with the term *simulated* if the system is based on chemical laws, i.e., it does not incorporate chemistry just as a metaphor. To avoid confusion we follow this convention and we avoid using the term artificial chemistry but rather use a more specific term *chemical reaction network*, which we employ to model our autonomous learners.

2.4 LEARNING IN CHEMISTRY

Even though, neural networks are widely used, the mathematical abstraction and network formalism detached them from their physical counterpart (their original inspiration). For instance, the calculation of a dot-product of inputs and weights, or a sigmoidal amplification do not address how a neural substrate physically represents values and functions. Physical neurons are electrical machines: synaptic voltage travels through ion channels and contributes to the membrane voltage, which spikes once it exceeds a constant threshold current. A model that deals with a neuron's physical realization to that fine detail is the spiking Hodgkin-Huxley neuron [67] and the related leaky integrate-and-fire neuron [141]. They are both driven by an electrical engineering formalism and incorporate terms such as resistance, capacitance, and Ohm's law.

Our goal has parallels to the leaky integrate-and-fire neuron model, but rather than describing mechanics of an existing neuron we aim to implement learning and adaptation in a chemical substrate, not necessarily the same as in its biological counterpart. Hence instead of voltage, the chemical models feed on molecular species and are driven by kinetic laws. Because of different premises and attributes, neural models cannot be translated directly to chemistry, but rather serve qualitatively to exemplify feedback control and adaptation.

The idea of neural network computation in chemical systems is not new. Several theoretical and experimental DNA-based models [29,32,65,66,85,109] have been proposed, however, these models succeeded to map to chemistry only the input-weight integration part (the forward-pass) of a single perceptron model [132]. The research in this area has mainly been limited to constructing logic gates and assembling them into circuits to compute custom Boolean functions. For instance, in a theoretical work Hjelmfelt *et*

al. [66] extended the cyclic enzyme system, introduced by Okamoto for metabolic regulation, to mimic the functionality of a McCulloch-Pitts neuron, with a special focus on thresholding. The weights, represented by catalysts, drive the reactions, which consume an input species with either low (0-value) or high (1-value) concentration. Kim *et al.* [85] modeled the states of neurons as the concentrations of RNA species and synapses as transcriptionally controlled DNA switches. More recently, Qian *et al.* [128] demonstrated an experimental implementation of linear threshold circuits by using DNA strand displacement seesaw gates and combined these to construct a Hopfield network. In all cases the existing work does not dwell on the learning aspects of chemical neural networks. The supervised learning is either not considered at all or performed by an external (non-chemical) system that computes the weights for a formal neural network, before converting these to molecular concentrations to serve as parameters for the chemical implementation [85, 128]. The main challenge is that the integration of the forward-pass, where the input species transform to the output, with the backward-pass, where a user's action affects the concentrations of species that define the system's functionality, requires reactions with complicated and timing-sensitive relations, such as positive and negative (catalytic) feedback loops.

Beyond neural-network approaches, Lakin proposed a schema for rudimentary learning within enzymatic chemistry [89], equivalent to non-negative least squares regression. In wet chemistry, rote learning of small decision trees was exhibited [121] (training by example), but true learning calls for generalization. Recently Chiang *et al.* [37] proposed a construction method for chemical reaction networks to represent reconfigurable logic functions. As opposed to our work, the reconfiguration is performed manually by controlling the concentrations of certain knob species. Wu *et al.* [161] surveyed a wide range of DNA-based logic gates, computing, and their applications in biotechnol-

ogy and biomedicine. In related work El-Samad and Khammash [45] explored feedback control in gene regulatory networks manifested in various dynamics, in particular bistability, mulistability, oscillations, and molecular switches. Jiang *et al.* [81] introduced the concept of a delay element used as a storage area for holding data in between each computation cycle. Moreover, a system based on gene regulatory networks embedded in *Escherichia coli* [52] demonstrated that even single-celled organisms could carry out associative learning.

Spiking neural P systems [76, 77] are related types of systems, which draw inspiration from neural network theory and incorporate membrane computing with a model of spiking neurons. Each neuron is wrapped in a membrane, where inter-neuron (intermembrane) communication is carried by the electrical impulses, called spikes. Some attempts were made to introduce learning to neural P systems [60], however, similarly to DNA-strand implementations, learning has not been autonomous either.

3

VALUE REPRESENTATION

In this chapter, we address the question of how to represent numerical values in chemistry. We present several novel methods on how to map `Boolean` or `Real` values into the concentrations of chemical species. We will later see how these representation approaches affect the design of binary and analog chemical perceptron models, including their learning performance and robustness.

A CRN, which we introduced in Section 2.3.1, can represent a formal variable by one or several species. In our chemical models we need to encode variables of two types: `Boolean` with values 0 and 1, and `Real` with values from $\mathbb{R}$. We transform variables to species in the systematic fashion as follows.

3.1 REAL VARIABLE

If the domain of a `Real` variable is positive, its value could directly correspond to the concentration of a species. For the signed real variables, a direct mapping to the concentrations of species would not work because the concentration cannot be negative. Note that the problem of representing negative numbers is equivalent to the problem of implementing the subtraction operation. Hence, if we restrict the model to positive numbers only, we could only add but not subtract numbers. That might be acceptable for some

models, but what if negative numbers cannot be avoided? How can we deal with that in chemistry?

A possible first approach is to introduce a special negative variant of each species and to extend pure addition-based chemical arithmetic with a subtraction operation wherein the complementary species annihilate when they occur simultaneously in a reactor (Figure 3.1(a)). This strategy maps each formal real variable p to two species $P^{\oplus}$ and $P^{\ominus}$, hence it is an instance of representation symmetry. Intuitively, after the complementary species (whose concentrations we wish to compare or subtract) annihilate, their original state is lost. If the goal is to repeat this comparison the system must also maintain backup copies $\overline{P}^{\oplus}$ and $\overline{P}^{\ominus}$ by consuming an externally provided fuel. After the comparison is completed the copies $\overline{P}^{\oplus}$ and $\overline{P}^{\ominus}$ are used to restore the original species. Because of the $P \to \overline{P} \to P$ reversibility, to prevent an infinite loop, we must precisely time the recovery phase and have a special species catalyzing (guarding) the comparison.

An improved symmetric strategy is to compare the concentrations of species indirectly by their impact on the concurrent reactions they catalyze, and so annihilation occurs at the level of complementary products. Let us consider a chemical system in which a substrate S is transformed to a product $P^{\oplus}$ or to $P^{\ominus}$, depending on the concentrations of two concurrent catalysts $E^{\oplus}$ and $E^{\ominus}$, as shown in Figure 3.1(b). Assuming the reaction rates of these two reactions are equal, the catalysts represent a *positive* number if and only if the final concentration of product $[P^{\oplus}]_{t\to\infty} > [P^{\ominus}]_{t\to\infty}$, which holds for $[E^{\oplus}]_0 > [E^{\ominus}]_0$, otherwise they represent a *negative* number. Since all products are derived from substrate S, $[P^{\oplus}]_t + [P^{\ominus}]_t + [S]_t \leq [S]_0$ and finally $[P^{\oplus}]_{t\to\infty} + [P^{\ominus}]_{t\to\infty} \leq [S]_0$.

What other approach could we take to implementing subtraction in chemistry? The general case may be difficult, but what if a qualitative comparison rather than precise subtraction suffices? Since the interpretation of positive and negative numbers is external
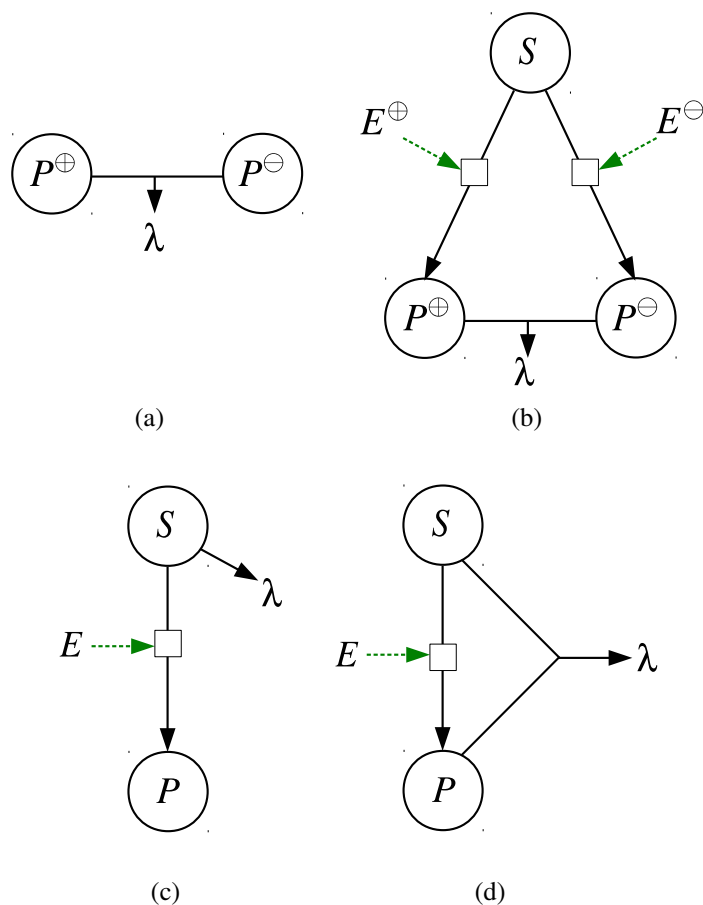
Figure 3.1: Implementations of positive and negative numbers in chemistry. (a-b) Symmetric approach using (a) annihilation of complementary species $P^\oplus$ and $P^\ominus$, or (b) competition of catalysts $E^\oplus$ and $E^\ominus$. (c-d) Asymmetric approach wherein a single catalyst E competes with (c) the substrate decay, or (d) annihilation of the substrate S and the product P. λ stands for no species, i.e., in a chemical implementation, an inert waste product.

(performed by us), the mapping from the concentration of catalyst(s) can be arbitrary. To eliminate the representation symmetry, we can keep just one catalyst E and one product P, but then all of the substrate S will eventually turn to product P regardless the rates or the non-zero concentration of the catalyst E. Therefore, we have no choice but to reintroduce a competition. Even though a negative catalyst is banned, we can still achieve a race if we introduce a decay of substrate $S \rightarrow \lambda$. Hence, the catalyst E must work against a

pressure, which is linear in the concentration of S (Figure 3.1(c)). The final concentration of product P after the experiment $[P]_{t\to\infty}$, depends on the rate of the decay reaction and the concentration of E, and is bounded by $[P]_{t\to\infty} < [S]_0$. To exploit this mechanism for the representation of real numbers, we set the threshold concentration Θ_P and Θ_E such that $[P]_{t\to\infty} > \Theta_P$ if and only if $[E]_0 > \Theta_E$. For a given initial substrate concentration $[S]_0$, the product concentration threshold $\Theta_P < [S]_0$, and reaction rates, we can determine the threshold concentration of catalyst Θ_E such that $[E]_0 > \Theta_E$ produces the concentration of product $[P]_t$ that is interpreted as *positive*, or if $[E]_0 \leq \Theta_E$ as *negative*. The relation between the concentration of catalyst $[E]_0$ and the final concentration of product $[P]_{t\to\infty}$, and therefore also between the thresholds Θ_E and Θ_P, is plotted in Figure 3.2(a).

An alternative version of the asymmetric comparison forces a catalyst E to compete against annihilation of substrate S and product P (Figure 3.1(d)). The relation between $[E]_0$ and $[P]_\infty$ is slightly different (Figure 3.2(b)), but it again acts as a monotonically increasing function. The initial concentration of substrate $[S]_0$ restricts the range of representable numbers in both situations, with or without symmetry. Note that only limited theoretical conclusions about the decay and annihilatory based asymmetric comparison can be drawn owing to the complexity of the underlying ODEs (see Appendix). However, in Chapter 8, after flattening the catalytic reactions to mass-action kinetics, we rigorously analyze the ODEs and derive the closed or approximative formulas.

3.2 BOOLEAN VARIABLE

Similarly to the real-valued case, a `Boolean` variable can be represented by either two-species encoding, which is an instance of symmetric representation, or one-species (asymmetric) encoding. The symmetric encoding maps a variable p into two species P^0 and P^1,
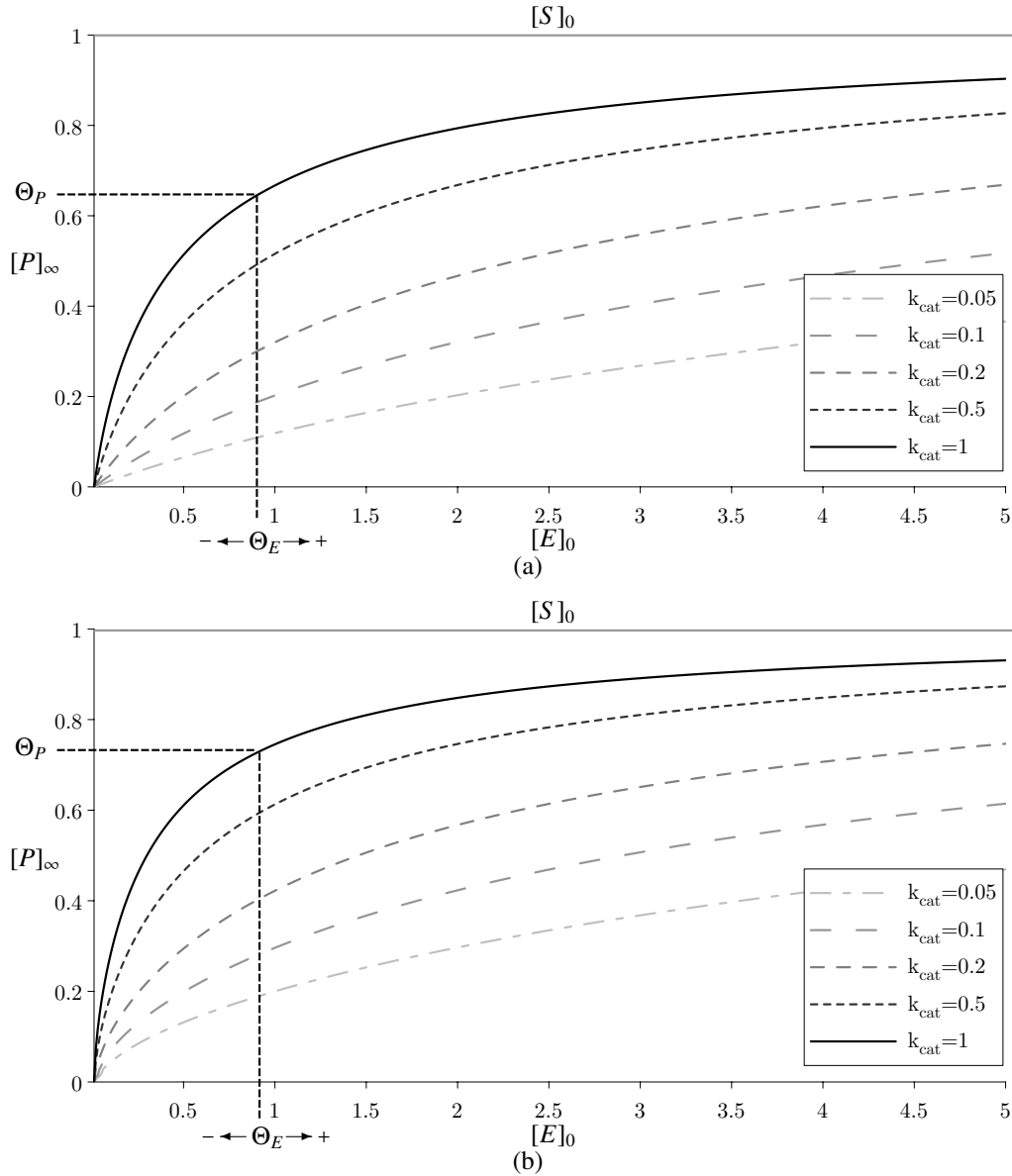
Figure 3.2: Relation between the concentration of catalyst $[E]_0$ and the final concentration of product $[P]_\infty$ for the asymmetric representation of real numbers by (a) decay of the substrate and (b) the annihilation of substrate and product using different k_{cat} rate constants and fixed $K_m = 0.05$ of the catalytic $S \xrightarrow{E} P$ reaction (Michaelis-Menten kinetics) using 0.01-step Runge-Kutta4 numerical integration. The rate of the substrate decay as well as the substrate-product annihilation is 1. For a given threshold product concentration Θ_P (y-axis) we can determine the associated catalyst threshold Θ_E (x-axis), so all concentrations of catalyst $[E]_0$ to the left of this threshold represent negative numbers, and all concentrations to the right represent positive numbers. The $[P]_\infty$ asymptotically reaches the initial concentration of the substrate $[S]_0 = 1$ (upper line) for $[E]_0 \rightarrow \infty$.

which are mutually exclusive. The non-zero concentration of P^0 denotes $p = 0$, analogously $[P^1]$ non-zero implies $p = 1$. If both variants P^0 and P^1 are simultaneously present in the tank, they annihilate very rapidly. The value is therefore interpreted as logic one if the concentration of P^1 is greater than the concentration of P^0, and logic zero otherwise, i.e., $[P^1] > [P^0]$. The analogous case for real-valued variable is shown in Figure 3.1(a).

An asymmetric approach utilizes just a single species, whose concentration represents both the logic zero and one value. To distinguish these two concentration regions we impose a concentration threshold Θ. Now we identify three types of thresholding: passive, semi-active, and active. The passive thresholding interprets the threshold externally by an outside observer, who compares the concentration of P to Θ, i.e., $[P] > \Theta$. The semi-active thresholding relies on injecting complementary species of given threshold concentration Θ to annihilate with P as performed, e.g., by Qian *et al.* [128]. This type of thresholding is conceptually similar to the passive tresholding, but instead of comparing the concentration with the threshold, we need to test whether the final concentration of species after annihilation is positive: $[P] > 0$. Moreover the semi-active thresholding is not repeatable without an experimenter intervention for the additional injection of complementary species.

3.3 ACTIVE THRESHOLDING

Active thresholding is a thresholding fully implemented in chemistry. It is a mechanism that amplifies the concentration to a specific upper value (representing logic one) if it exceeds the threshold, or reduces it to a lower value (representing logic zero) otherwise. The value interpretation distinguishes between two concentration levels: $[P] = P_{low}$ as the formal zero and $[P] = P_{high}$ as the formal one. The problem of active thresholding can

be effectively reduced to an implementation of a bistable regulator.

Wilhelm [159] proposed the *smallest chemical reaction system with bistability* using four reactions:

$$S + Y \xrightarrow{k_1} 2X$$

$$2X \xrightarrow{k_2} X + Y$$

$$X + Y \xrightarrow{k_3} Y + P$$

$$X \xrightarrow{k_4} P$$
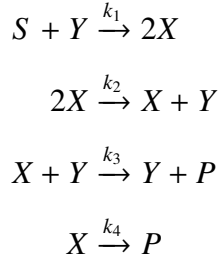
with two species X and Y, energy source S, and an inert product P. Since the concentration of S is constant and P is practically a waste, we can discard them from the reaction set and obtain a system with two species X and Y only. The system has three equilibrium states $\bar{x}_1 = 0$, $\bar{y}_1 = 0$, $\bar{x}_2 = (k_1 - \sqrt{k_1 D})/2k_3$, $\bar{y}_2 = \bar{x}_2^2/k_1$, $\bar{x}_3 = (k_1 + \sqrt{k_1 D})/2k_3$, $\bar{y}_3 = \bar{x}_3^2/k_1$ with discriminant $D = k_1 - 4k_3 k_4$. The first (lower value) and the third (upper value) solutions are locally stable, the second (threshold value) is unstable, hence if it is perturbed upwards, the system travels to the upper value; if it is perturbed downwards, it settles to $(0, 0)$ (Figure 3.3).

Now we can map the upper value P_{high} (logic one) to the upper stable point $\bar{y}_3$, the lower value P_{low} (logic zero) to the lower stable point $\bar{y}_1 = 0$, and the threshold Θ to the unstable point $\bar{y}_2$. Note that besides the species Y, there is an auxiliary species X undergoing qualitatively similar bistable behavior than Y. Since the thresholding of the species X is not utilized, we ignore and consider it a waste.
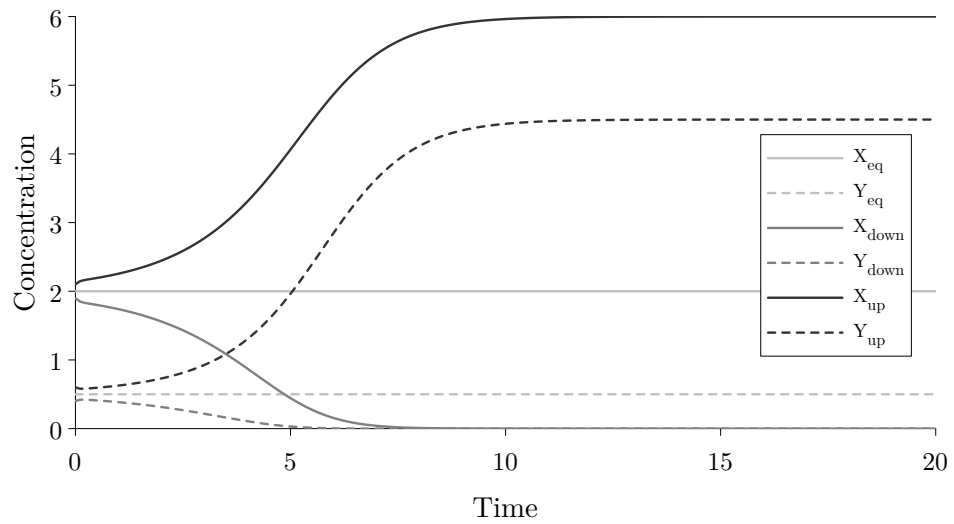
Figure 3.3: The original minimal bistable system introduced by Wilhelm [159] with three equilibrium states, two of which, lower and upper values, are stable, and the middle one (the threshold) is unstable. The figure shows concentration paths for the system perturbed in an upper direction and a lower direction.

4

BINARY CHEMICAL PERCEPTRON

In this chapter we will describe an implementation of the two-input binary CRN percep-
tron, an autonomous chemical learning system, which we call a *binary chemical percep-
tron*. It is the first chemical model capable of autonomous learning, that is the learning
fully implemented in simulated chemistry without any external help. We want to em-
phasize that many ways to approach this problem exist. Here we present four models,
the *Weight-Loop Perceptron* (WLP), the *Weight-Race Perceptron* (WRP), the *Asymmetric
Signal Perceptron* (ASP) and its thresholded version (TASP). These represent fundamen-
tal techniques for implementing the input-weight integration and learning in chemistry.
This work has been published in parts in [21, 22].

The WLP and the WRP adapt a symmetric representation of values and learn de-
sired outputs. The WLP's input-weight integration is based on a direct transformation of
weights to the output, and reconstructing them after the processing from back-up copies
forming a weight loop. It is the largest system of all presented and the only one that
requires inhibition. The WRP improves the WLP by changing the roles of the inputs
and weights—it applies indirect comparison of weights by letting them race on the input-
to-output reactions as catalysts. Supervised learning in both cases is triggered by a dis-
crepancy between the desired and the actual output species, where the concentration of
the desired output is additively or subtractively combined with the concentrations of the

weights. The output is interpreted by the concentration of complementary zero and one output species.

The WLP and the WRP learn perfectly all 14 linearly-separable logic functions after 200 learning iterations (100% success rate). They are also robust to perturbations of rate constants that substantially alleviate reaction-timing restrictions for real chemical implementations. Overall, the WLP and the WRP are proof-of-concept models for a chemical perceptron, however, the number of reactions and their complexity make them impractical for wet chemistry implementation. The underlying cause of that complexity is a crucial characteristic they share—a representation symmetry of the species that encode the formal variables. Namely, all real-valued variables require a positive and a negative variant of a species; and, similarly, two species, the zero and one variants, represent each binary variable.

The follow-up model of the ASP aims to simplify and reduce the number of reactions, such that an implementation in wet chemistry becomes possible. The main improvement is the abolition of representation symmetry, which reduces the number of reactions by half. The ASP contains no inhibition and uses at most one catalyst per reaction. Furthermore, the ASP introduces a special species, the input clock signal, that is provided alongside with the regular input. The ASP determines the output value by thresholding, as opposed to a comparison of two output species. The thresholding is either imposed by an external observer (passive thresholding), or is implemented fully in chemistry (active thresholding). A variant of the ASP with active thresholding (TASP) can support modularization and cascading of multiple perceptrons.

The learning mechanism has been revised and the ASP introduces adaptation by the biologically more plausible reinforcement method [80, 147]. More specifically, we train the ASP by injecting a penalty signal when it produces an incorrect output. The major

saving in design is offset by reduced robustness with respect to variation of kinetic rates, which is, however, still sufficiently high to mitigate the difficulties of precise reaction timing. On the other hand, the ASP maintains the high performance of the WLP and the WRP: it learns all 14 linearly-separable binary functions with a $99.3 - 99.99\%$ success rate.

All models are compatible with Michaelis-Menten [38, 108], and the ASP also with pure mass-action kinetics [13, 49]. Their reactions and rates provide a universal description that all chemists understand and consequently can translate to the implementation substrate of their choice, such as DNA hybridization [140, 166], or deoxyribozymes [95, 143, 145].

Each chemical perceptron can function in two modes: *input-weight integration mode* and *learning mode*. In the input-weight integration mode, the perceptron acts like a logic gate; it takes two inputs and produces an output by pairing inputs with weights and combining their contributions. The second learning mode is built on top of the input-weight integration and is triggered either by the desired-output molecules for the WLP and the WRP, or the penalty signal for the ASP.

4.1 BASIC SPECIES

Here we describe the core species used in the WLP, WRP, and ASP design, highlighting the differences resulting from adopted symmetric and asymmetric strategies for the representation of real and Boolean values (Chapter 3).

4.1.1 Input Species and the Clocked Representation

The two-input formal binary perceptron introduced in Section 2.1.1 accepts four possible inputs $(x_1, x_2) \in \{(0, 0), (1, 0), (0, 1), (1, 1)\}$. In our CRN we represent the presentation of

an input as the injection of a molecular species into the reaction chamber, such that each formal variable with its associated value maps to one or several molecular species.

The WLP and WRP designs use a straightforward domain enumeration (a symmetric approach) wherein each binary variable requires two species—one for the value 0 and one for the value 1, marked with a superscript. That is, the assignment $x_1 = 0$ translates into the injection of species X_1^0, and $x_1 = 1$ into the injection of species X_1^1. Analogously, the cases $x_2 = 0$ and $x_2 = 1$ are represented as X_2^0 and X_2^1, respectively. Therefore, the presence rather than the precise concentration of the input species represents a value. The input concentration is, however, not arbitrary and must reflect a specific scale, design, and it must be sufficient to feed inner reactions, because the input species are sole fuel for the WRP and the ASP. Note that the injections are consistent, so the complementary input species never occur together.

To reduce the number of input species and therefore also the number of reactions, we need to drop the 0/1 representation symmetry. A naive way would be to discard the zero-value species and provide input to a chemical perceptron only if it formally is the value one. The drawback here is that the input pair $(0, 0)$ would be represented as nothing. That is, a chemical perceptron would not know whether and when to produce the output for the input $(0, 0)$. We argue that a true implementation of a chemical logic gate must not treat the zero-valued input as an aberration, even if it is commonly done so in the biochemical computing literature. For instance, chemical systems of the form $X_1 + X_2 \rightarrow Y$ are commonly said to represent AND gates; because a measurable output Y is produced only for a simultaneous presence of both reactants, such systems should rather be called *two-signal detectors*.

The standard solution to deal with the zero-value problem, widely used in digital system design in electrical engineering, is to introduce a clock signal, provided alongside

Table 4.1: Representation of four binary input pairs by chemical species for the WLP, the WRP, the ASP, and a minimal representation that neglects the input $(0, 0)$.

Binary input	WLP/WRP	ASP	Minimal
$(x_1, x_2) = (0, 0)$	$\{X_1^0, X_2^0\}$	$\{S_{in}\}$	$\emptyset$
$(x_1, x_2) = (1, 0)$	$\{X_1^1, X_2^0\}$	$\{S_{in}, X_1\}$	$\{X_1\}$
$(x_1, x_2) = (0, 1)$	$\{X_1^0, X_2^1\}$	$\{S_{in}, X_2\}$	$\{X_2\}$
$(x_1, x_2) = (1, 1)$	$\{X_1^1, X_2^1\}$	$\{S_{in}, X_1, X_2\}$	$\{X_1, X_2\}$

the regular input. Even though this special signal, which we shall denote S_{in}, is, strictly speaking, required only for the input $(0, 0)$, we employ it for all input cases. The reason is the overall consistency and the functioning of the ASP. Since the goal is to imitate the weight sum $y = w_0 + w_1 x_1 + w_2 x_2$ we can consider the clock signal S_{in} the constant-one coefficient (or the constant input $x_0 = 1$) of the bias weight w_0, and so each weight has its own input species, and S_{in} always accompanies the regular input X_1 and X_2 (Table 4.1). This approach simplifies the design and makes the clock signal with the bias processing of the ASP conceptually independent from the X_1 and X_2 reactions, as we shall discuss in Section 4.2.

4.1.2 Output Species and Output Interpretation

For the output interpretation, we must carry out a reverse translation: we map the concentrations of the designated output species to the formal binary variable y. The WLP and the WRP have two complementary output species, Y^0 and Y^1, which are mutually exclusive, so if they occur simultaneously, they annihilate. We interpret the output as one if the concentration of Y^1 is greater than the concentration of Y^0, and zero otherwise, i.e., $y = [Y^1] > [Y^0]$, which we call passive thresholding. The ASP contains only one output species Y; therefore, to distinguish between zero and one output, we impose a threshold concentration Θ, and externally interpret the output as $y = [Y] > \Theta$. Another version

of the ASP with internal thresholding distinguishes between two concentration levels:

$[Y] = 0$ as the formal zero and $[Y] = 1.5$ as the formal one.

Table 4.2: Species of (a) the WLP, (b) the WRP, (c) the ASP. The species are divided into groups according to their purpose and functional characteristics. Note that in addition to the species of the ASP, the TASP also uses an auxiliary output species Y_{aux}.

(a) WLP

Function	Species
Inputs	$X_1^0, X_1^1, X_2^0, X_2^1$
Outputs	Y^0, Y^1
Weights	$W_0^\oplus, W_0^\ominus, W_1^\oplus, W_1^\ominus, W_2^\oplus, W_2^\ominus$
Desired outputs	D^0, D^1
Processed weights	$\overline{W}_0^\oplus, \overline{W}_0^\ominus, \overline{W}_1^\oplus, \overline{W}_1^\ominus, \overline{W}_2^\oplus, \overline{W}_2^\ominus$
Energy source	E
Total	21

(b) WRP

Function	Species
Inputs	$X_1^0, X_1^1, X_2^0, X_2^1$
Outputs	Y^0, Y^1
Weights	$W_0^\oplus, W_0^\ominus, W_1^\oplus, W_1^\ominus, W_2^\oplus, W_2^\ominus$
Desired outputs	D^0, D^1
Total	14

(c) ASP

Function	Species
Inputs	X_1, X_2
Output	Y
Weights	W_0, W_1, W_2
Penalty signal	P
Input (clock) signal	S_{in}
Weight changers	$W^\ominus, W^\oplus, W_1^\ominus, W_2^\ominus$
Total	12

4.1.3 Weight Species

All chemical perceptrons include the weight species (Table 4.2), which hold the perceptron's state and define its functionality. After each learning iteration the perceptron needs to recover to its steady state. Only the weight species form the persistent state, hence all other species not consumed during the chemical computation must be flushed or removed by a clean-up reaction such as decay. Furthermore, there are WLP-, WRP- and ASP-specific species groups, the purpose of which will be explained later.

4.2   INPUT-WEIGHT INTEGRATION

To implement the input-weight integration of the two-input perceptron, we must cover the four weight sums from Table 2.1. Since the representation of negative numbers and the subtraction operation cannot be avoided, we demonstrate how the symmetric and asymmetric approaches to the chemical representation of real numbers affect the design of the WLP, the WRP and the ASP.

4.2.1   Weight-Loop Perceptron

The WLP, which consists of 21 species (Table 4.2(a)), follows the formal perceptron definition from Section 2.1.1 in a straightforward manner. More precisely, the WLP computes the weight sum directly by transforming weights W into output species Y. The problem is that the weights encode the state of the perceptron, so their concentration must be preserved. Therefore, besides Y species, the perceptron must create also back-up copies of the weights $\overline{W}$. The perceptron can then restore its weights after the output production is over. A reaction $W_i \rightarrow \overline{W}_i + Y$ followed by $\overline{W}_i \rightarrow W_i$ would break the mass-conservation law, so the perceptron needs to consume a fuel, species E, that is provided to the system at constant concentration 1. From a functional perspective, the perceptron sequentially processes an input, produces an output, recovers weights, and finally performs a clean-up (Figure 4.1).

The perceptron starts working when inputs X_1 and X_2 are injected into the system. It processes the weight W_1 on input X_1^1, the weight W_2 on input X_2^1, and the weight W_0, in parallel, producing Y^0 and Y^1 molecules. Species X_1^1, formally encoding $x_1 = 1$, catalyzes $\oplus$ and $\ominus$ versions of reaction $W_1 + E \rightarrow \overline{W}_1 + Y$. Similarly, species X_2^1, which represents $x_2 = 1$, catalyzes $W_2 + E \rightarrow \overline{W}_2 + Y$. Since the weight W_0 always contributes to the sum
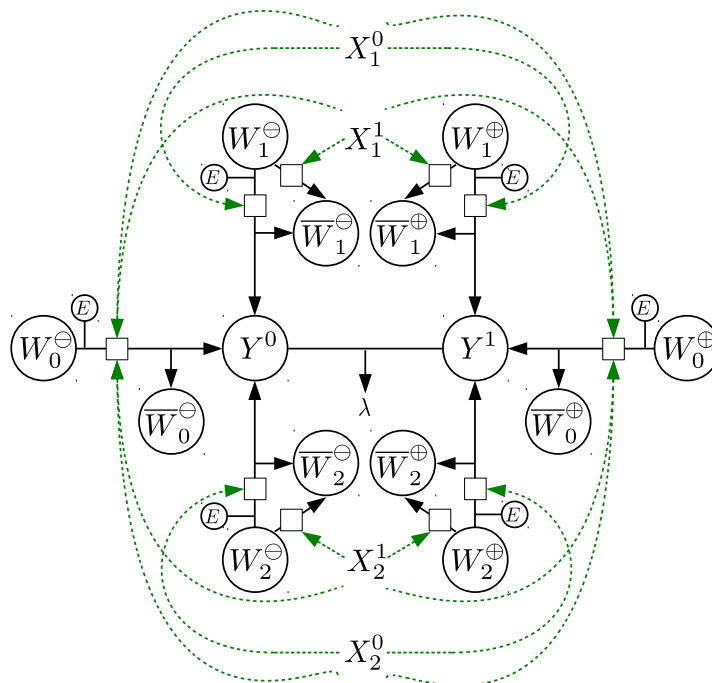
Figure 4.1: The WLP's reactions employed in the input-weight integration. The input species X trigger (catalyze) a reaction $W + E \rightarrow \overline{W} + Y$, which consumes weight W and fuel E, and produces output Y and a back-up copy of the weight $\overline{W}$. After the output is produced, the WLP recovers the weights by reactions $\overline{W} \rightarrow W$ (not shown here). For simplification neither the input decay reactions are present in the plot. Nodes represent species, solid lines are reactions, dashed lines are catalysts, and λ is no or inert species. The horizontal symmetry is due to the existence of two perceptron's inputs; the vertical one is the representation symmetry globally spread across the WLP design.

regardless of an input, each of the possible inputs X_1^0, X_1^1, X_2^0, and X_2^1 catalyzes $W_0 + E \rightarrow \overline{W}_0 + Y$. In order to determine whether the total concentration of Ys is above or below the zero threshold, Y^0 annihilates with Y^1. If there are more Y^0 molecules at the end, the output is 0, otherwise 1.

The weights could alternate between the normal version W and the processed version $\overline{W}$, each time consuming a fuel E and producing new Y molecules. To prevent a continuous cycling of the weights, the WLP must ensure that there is no input present before it rolls the weights back. That is, the input species must decay, and the processed weights roll back only when substantial amounts of inputs are gone, i.e., inputs act as inhibitors

on $\overline{W} \to W$ reactions. Since the system is open and weights W can switch reversibly to $\overline{W}$, consuming a fuel E provided from outside, a potential infinite loop might emerge in which the concentration of Y molecules increase without bound. The correct timing of phases is crucial for avoiding this problem. The output molecules Y are removed from the system by a decay.

Table 4.3(a) presents the full set of reactions with associated catalysts and inhibitors. Note that each reaction with multiple independent catalysts (group 1) or inhibitors (group 4) is a compressed version of the reactions each using just one input species $X_1^0, X_1^1, X_2^0, X_2^1$ as a catalyst or an inhibitor. Therefore, the group 1 expands to 8 reactions, the group 4 to 24 reactions, and overall the WLP requires 54 rather than 34 reactions shown in the table.

4.2.2 Weight-Race Perceptron

The functioning of the WLP is based on rather conservatively-designed phases working in a sequence. This approach works well since the WLP directly implements the input-weight integration routine of the formal binary perceptron. Nevertheless, the idea of direct calculation of the weight sum and recovering the original state seems unnecessarily cumbersome for a chemical system, resulting in a large number of species and reactions.

The WRP, consisting of 14 species and 30 reactions, improves the weight-loop model by switching the chemical roles of inputs and weights. Instead of having inputs catalyzing a transformation of weights to a weight sum, which determines an output, weights catalyze the input-to-output reactions as presented in Figure 4.2. Input species X_1 and X_2 are transformed directly to Y^0 or Y^1 depending on the sign of currently present weights. Thus, the WRP does not compare weights directly, but lets them compete on input-to-output reactions as catalysts, so it basically implements a rate (derivation-based) comparison. For this to work, species Y^0 must annihilate with Y^1 quickly, racing must be simultaneous,
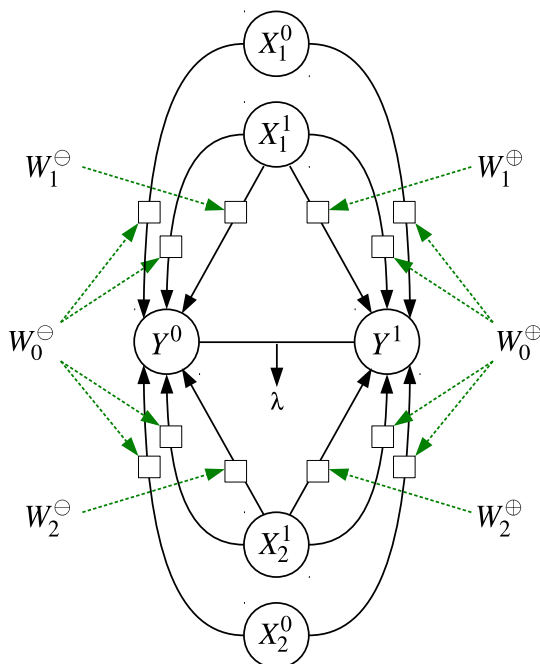
Figure 4.2: The WRP's reactions employed in the input weight integration. The weights W catalyze (compete on) the input-to-output reactions $X \rightarrow Y$. Nodes represent species, solid lines are reactions, dashed lines are catalysts, and λ is no or inert species. The horizontal symmetry is due to the existence of two perceptron's inputs; the vertical one is the representation symmetry globally spread across the WRP design.

and the rate functions must have a similar shape.

This is an instance of the second symmetric approach (Figure 3.1(b)), in which the sum of positive and negative numbers is calculated indirectly by the impact of a catalytic species racing on a shared substrate. Whether the WRP produces more Y^1 or Y^0 is determined by the cumulative strength of all $\oplus$ over the $\ominus$ weights, which includes both the weight concentrations and the catalytic reaction rates.

The weight $W_1^\oplus$ (or $W_1^\ominus$) catalyses solely the reaction $X_1^1 \rightarrow Y^1$ (or $X_1^1 \rightarrow Y^0$), analogously for $W_2^\oplus$ and $W_2^\ominus$. Since the bias weights $W_0^\oplus$ and $W_0^\ominus$ are always active, they drive all input-to-output reactions. Note that at this point the formal weight values stop matching the concentrations of the weight species, owing to the non-linear nature of a catalytic

race. Weights catalyze the reactions concurrently, so the one with the highest concentration consumes the largest portion of an input and has therefore the highest contribution in an output. Analogously to the weight-loop model, an annihilation of Y^1 and Y^0 decides whether the concentration of Ys is above or below the zero threshold.

If the WRP is supposed to treat all weights equally, it must ensure that the weight race is fair. Following the formal perceptron definition, the contribution of weights in the sum must be uniform, meaning there is no preference among weights. Besides the weight concentration, the reaction rate constants determine the actual speed of the input consumption.

Now, if all weights have the same sign, then it does not matter what rate constants are set, so we can let the qualitative state of the perceptron be $W_0^\oplus$, $W_1^\ominus$ and $W_2^\ominus$ (Figure 4.3). For inputs X_1^0 and X_2^0 only the weight $W_0^\oplus$ is active, so there is no racing. The weight $W_0^\oplus$ competes with $W_1^\ominus$ and $W_2^\ominus$ for inputs X_1^1 and X_2^1; however, $W_0^\oplus$ is privileged since it consumes X_1^1 and X_2^1 at the same time. Note that $W_1^\ominus$ consumes just X_1^1 and $W_2^\ominus$ just X_2^1. To fix this issue we have to penalize $W_0^\oplus$ by setting the rate constants of the reactions $X_1^1 \rightarrow Y^1$ and $X_2^1 \rightarrow Y^1$ both catalyzed by $W_0^\oplus$ to δ, and those catalyzed by $W_1^\ominus$ and $W_2^\ominus$ to 2δ. As a result, the contribution preference of the weights is balanced.

The new problem emerges for inputs X_1^1 and X_2^0, where $W_0^\oplus$ drives two, but $W_1^\ominus$ just one reaction. Even if the X_1^1 reactions follow the two-to-one rate constant ratio, whatever constant assigned to the reaction $X_2^0 \rightarrow Y^1$ catalyzed by $W_0^\oplus$ will result in an unfair advantage for $W_0^\oplus$, since eventually all X_2^0 molecules will change to Y^1. To balance the preference of $W_0^\oplus$ we need to introduce a decay of X_2^0, such that exactly two thirds are taken away. In order to do that, $W_0^\oplus$ must catalyze not just $X_2^0 \rightarrow Y^1$ with the rate constant δ, but also the decay $X_2^0 \rightarrow \lambda$ with the rate constant 2δ. That is the reason why the reaction set of the WRP contains a decay of the input species X_1^0 and X_2^0 (Table 4.3(b), Group 2).
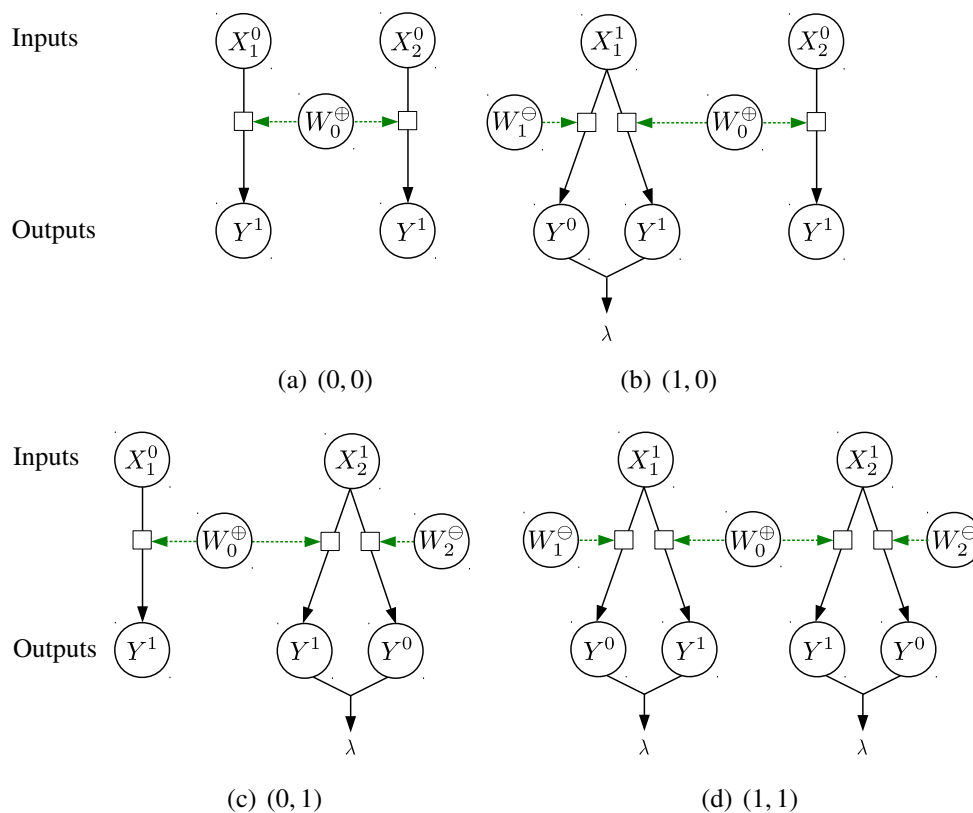
Figure 4.3: The input-output precessing of the WRP for positive W_0 and negative W_1 and W_2 weights and 4 input pairs.

Based on our simulations, it however turns out that the WRP can perform well regardless of the preference among weights because it can compensate a certain degree of unfairness in the race by non-uniform weight adaptation. Even though a bias exists, we can always find such a concentration of weight species that the overall production of Y^1 over Y^0 will follow a prescribed binary function profile.

The WRP is substantially simpler compared to the WLP. It contains 30 reactions without any inhibition. Unlike the WLP, the system does not need any externally supplied fuel species. In fact, the input species adapts this role, so they are essentially an information and energy source.

Table 4.3: The full reaction list for (a) the WLP and (b) the WRP. Reactions are divided into groups according to common functional characteristics. The symbol $\forall X$ denotes all inputs species $X_1^0, X_1^1, X_2^0, X_2^1$.

(a)

#	Reaction	Catalysts	Inhibitors
1	$W_0^\oplus + E \rightarrow \overline{W}_0^\oplus + Y^1$	$\forall X$	
	$W_0^\ominus + E \rightarrow \overline{W}_0^\ominus + Y^0$	$\forall X$	
2	$W_1^\oplus + E \rightarrow \overline{W}_1^\oplus + Y^1$	X_1^1	
	$W_1^\ominus + E \rightarrow \overline{W}_1^\ominus + Y^0$	X_1^1	
	$W_2^\oplus + E \rightarrow \overline{W}_2^\oplus + Y^1$	X_2^1	
	$W_2^\ominus + E \rightarrow \overline{W}_2^\ominus + Y^0$	X_2^1	
3	$W_1^\oplus \rightarrow \overline{W}_1^\oplus$	X_1^0	
	$W_1^\ominus \rightarrow \overline{W}_1^\ominus$	X_1^0	
	$W_2^\oplus \rightarrow \overline{W}_2^\oplus$	X_2^0	
	$W_2^\ominus \rightarrow \overline{W}_2^\ominus$	X_2^0	
4	$\overline{W}_0^\oplus \rightarrow W_0^\oplus$		$\forall X$
	$\overline{W}_0^\ominus \rightarrow W_0^\ominus$		$\forall X$
	$\overline{W}_1^\oplus \rightarrow W_1^\oplus$		$\forall X$
	$\overline{W}_1^\ominus \rightarrow W_1^\ominus$		$\forall X$
	$\overline{W}_2^\oplus \rightarrow W_2^\oplus$		$\forall X$
	$\overline{W}_2^\ominus \rightarrow W_2^\ominus$		$\forall X$
5	$W_0^\oplus + W_0^\ominus \rightarrow \lambda$		
	$W_1^\oplus + W_1^\ominus \rightarrow \lambda$		
	$W_2^\oplus + W_2^\ominus \rightarrow \lambda$		
6	$Y^0 + Y^1 \rightarrow \lambda$		
7	$X_1^0 \rightarrow \lambda$		
	$X_1^1 \rightarrow \lambda$		
	$X_2^0 \rightarrow \lambda$		
	$X_2^1 \rightarrow \lambda$		
8	$Y^0 \rightarrow \lambda$		
	$Y^1 \rightarrow \lambda$		
9	$D^0 \rightarrow \lambda$		
	$D^1 \rightarrow \lambda$		
10	$D^0 \rightarrow W_0^\ominus$	Y^1	
	$D^1 \rightarrow W_0^\oplus$	Y^0	
11	$D^0 \rightarrow W_1^\ominus$	Y^1, X_1^1 (AND)	
	$D^0 \rightarrow W_2^\ominus$	Y^1, X_2^1 (AND)	
	$D^1 \rightarrow W_1^\oplus$	Y^0, X_1^1 (AND)	
	$D^1 \rightarrow W_2^\oplus$	Y^0, X_2^1 (AND)	

(b)

#	Reaction	Catalysts
1	$X_1^0 \rightarrow Y^1$	$W_0^\oplus$
	$X_1^0 \rightarrow Y^0$	$W_0^\ominus$
	$X_2^0 \rightarrow Y^1$	$W_0^\oplus$
	$X_2^0 \rightarrow Y^0$	$W_0^\ominus$
	$X_1^1 \rightarrow Y^1$	$W_0^\oplus$
	$X_1^1 \rightarrow Y^0$	$W_0^\ominus$
	$X_2^1 \rightarrow Y^1$	$W_0^\oplus$
	$X_2^1 \rightarrow Y^0$	$W_0^\ominus$
2	$X_1^0 \rightarrow \lambda$	$W_0^\oplus$
	$X_1^0 \rightarrow \lambda$	$W_0^\ominus$
	$X_2^0 \rightarrow \lambda$	$W_0^\oplus$
	$X_2^0 \rightarrow \lambda$	$W_0^\ominus$
3	$X_1^1 \rightarrow Y^1$	$W_1^\oplus$
	$X_1^1 \rightarrow Y^0$	$W_1^\ominus$
	$X_2^1 \rightarrow Y^1$	$W_2^\oplus$
	$X_2^1 \rightarrow Y^0$	$W_2^\ominus$
4	$W_0^\oplus + W_0^\ominus \rightarrow \lambda$	
	$W_1^\oplus + W_1^\ominus \rightarrow \lambda$	
	$W_2^\oplus + W_2^\ominus \rightarrow \lambda$	
5	$Y^0 + Y^1 \rightarrow \lambda$	
6	$Y^0 \rightarrow \lambda$	
	$Y^1 \rightarrow \lambda$	
7	$D^0 \rightarrow \lambda$	
	$D^1 \rightarrow \lambda$	
8	$D^0 \rightarrow W_0^\ominus$	Y^1
	$D^1 \rightarrow W_0^\oplus$	Y^0
9	$D^0 \rightarrow W_1^\ominus$	Y^1, X_1^1 (AND)
	$D^0 \rightarrow W_2^\ominus$	Y^1, X_2^1 (AND)
	$D^1 \rightarrow W_1^\oplus$	Y^0, X_1^1 (AND)
	$D^1 \rightarrow W_2^\oplus$	Y^0, X_2^1 (AND)

Table 4.4: The full reaction list for (a) the ASP; (b) the extra thresholding reactions of the TASP. Reactions are divided into groups according to common functional characteristics.

(a)			(b)	
#	Reaction	Catalyst	#	Reaction
1	$S_{in} + Y \to \lambda$		12	$Y_{aux} \to 2Y$
2	$S_{in} \to Y$	W_0	13	$2Y \to Y + Y_{aux}$
3	$X_1 + Y \to \lambda$		14	$Y + Y_{aux} \to Y_{aux}$
	$X_2 + Y \to \lambda$		15	$Y \to \lambda$
4	$X_1 \to Y$	W_1		
	$X_2 \to Y$	W_2		
5	$P \to W^{\oplus}$			
6	$P \to W^{\ominus}$	Y		
7	$W^{\ominus} + W_0 \to \lambda$			
8	$W^{\oplus} \to W_0$			
9	$W^{\ominus} \to W_1^{\ominus}$	X_1		
	$W^{\ominus} \to W_2^{\ominus}$	X_2		
10	$W_1 + W_1^{\ominus} \to \lambda$			
	$W_2 + W_2^{\ominus} \to \lambda$			
11	$W^{\oplus} \to W_1$	X_1		
	$W^{\oplus} \to W_2$	X_2		

4.2.3 Asymmetric Signal Perceptron

In the design of the ASP we further extend the idea of an unfair race embedded in the WRP design. It exploits the reaction rate setting to ensure that the contribution of weights allows the representation of negative numbers and subtraction, but at the same time avoids the $\oplus$ vs. $\ominus$ symmetry of weights. The set of input species in the ASP shrinks from $\{X_1^0, X_1^1, X_2^0, X_2^1\}$ to the three species $\{X_1, X_2, S_{in}\}$ (Table 4.1). The input (clock) signal S_{in}, primarily needed for the input pair $(0, 0)$, when neither X_1 nor X_2 is injected, can be elegantly incorporated to the rest of input pairs to serve an additional purpose. Because the bias weight is always included in the weight sum regardless of the input (see Table 2.1), we can extract the bias processing part and design the ASP such that the input signal S_{in} will also be the weight-species W_0 specific substrate. Therefore, the input signal S_{in}

is always injected and it accompanies the regular input species X_1 and X_2 (if provided). This is a simpler alternative than hooking the W_0 species to all possible inputs as in the WRP. The presence of event signals, the input signal and the penalty signal (introduced in Section 4.3), is another prominent feature of the ASP.

Now, using the asymmetric representation of numbers by a single catalyst (Figure 3.1(c)), we obtain three weight species W_0, W_1, and W_2 as opposed to $W_0^{\oplus}$, $W_0^{\ominus}$, $W_1^{\oplus}$, $W_1^{\ominus}$, $W_2^{\oplus}$, and $W_2^{\ominus}$ as required by the WLP/WRP. Owing to the introduction of the input signal S_{in}, each weight species consumes its own input and adds a portion to the global output species Y, as shown in Figure 4.4(a). By imposing a certain threshold concentration Θ we create a system in which each weight species races with its private decay, and consequently the weight concentrations can represent both positive and negative numbers.

On the other hand, this system lacks cross-weight racing because of the disconnection of weight impacts. More precisely, a small concentration of one weight resulting in a stronger weight-specific decay would never affect the contribution of a different weight to the global output. Since no arrow points out of the species Y, once produced, Y cannot be consumed, therefore the system is additive. The output concentration $[Y]$ would consist of three portions $[Y]_{S_{in}}$, $[Y]_{X_1}$, and $[Y]_{X_2}$, corresponding to the output produced from the input species S_{in}, X_1, and X_2. Because the weights do not influence one another (their contributions are strictly additive), the output for the formal inputs $(0,0)$, $(1,0)$, $(0,1)$, and $(1,1)$ would be $[Y]_{(0,0)} = [Y]_{S_{in}}$, $[Y]_{(1,0)} = [Y]_{S_{in}} + [Y]_{X_1}$, $[Y]_{(0,1)} = [Y]_{S_{in}} + [Y]_{X_2}$, and $[Y]_{(1,1)} = [Y]_{S_{in}} + [Y]_{X_1} + [Y]_{X_2}$ consecutively. Now, since the threshold or the output interpretation is the same for all inputs, the output concentrations could not represent binary functions that are non-monotonously increasing, such as `NAND` or `NOR` for any weight concentrations. For instance, `NAND` requires $[Y]_{(1,1)} = [Y]_{S_{in}} + [Y]_{X_1} + [Y]_{X_2} \leq \Theta$, and $[Y]_{(0,0)} = [Y]_{S_{in}} > \Theta$, which is not possible. Hence, instead of one global race embracing
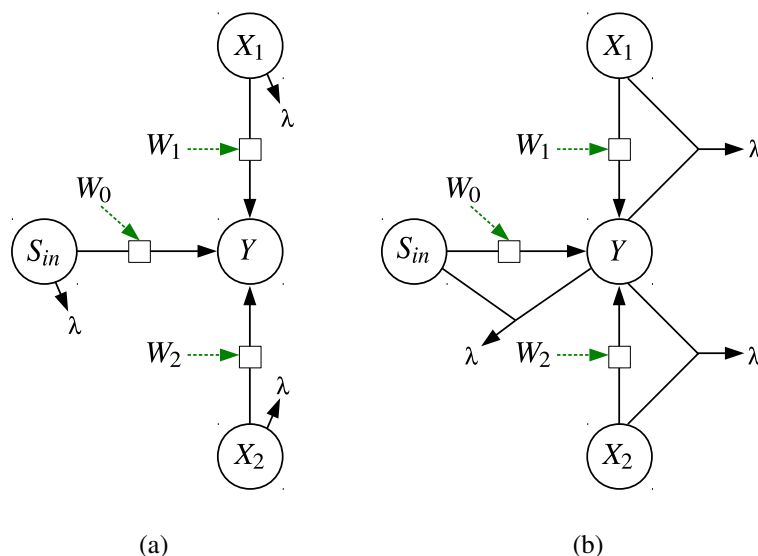
Figure 4.4: The initial (a) and the final (b) version of the ASP's reactions employed in input-weight integration incorporating an asymmetric representation of real values (and subtraction). The initial version (a) using decay of inputs fails to model a general linearly separable binary function due to pure additive contributions of input-weight branches. The final version (b) overcomes this problem and enforces cross-weight competition by introducing the annihilation of inputs and output. Nodes represent species, solid lines are reactions, dashed lines are catalyses, and λ is nothing, or an inert species.

all weights, we would end up having three independent races with additive contributions.

What could we do to impose a cross-weight global race? As we mentioned earlier, there is no negative pressure, such as decay, that would interlink the products of weight catalyses. In other words, we need the reaction arrows to head not just into but also out of the output species Y. Naively introducing a decay of the output species $Y \rightarrow \lambda$ would not work because a negative pressure or consumption must be conditional on the input type. Depending on the presence of S_{in}, X_1, and X_2, a certain part of negative pressure must be controlled (turned on or off). To address that, we replace the original asymmetric building block with a version using an annihilation of the substrate (input) and product, instead of the decay (Figure 3.1(d)), and thus obtain the system shown in Figure 4.4(b), which can qualitatively imitate the two-input perceptron. As the concentration of weights

increases, the output increases as well and asymptotically reaches the total amount of input injected. The upper bound for the final output's concentration is therefore $[Y] \leq [X_1]_0 + [X_2]_0 + [S_{in}]_0$, which holds if both inputs X_1 and X_2 are injected. For the input $(0, 0)$ only the clock signal S_{in} penetrates the system, therefore the upper bound for this case $[Y] \leq [S_{in}]_0$. Since we compare the output concentration with the same threshold Θ for all four possible inputs, $\Theta < [Y] \leq [S_{in}]_0 \leq [X_1]_0 + [X_2]_0 + [S_{in}]_0$. We set the threshold concentration Θ to 0.5, which allows enough maneuverability in both positive and negative region.

4.2.4 Thresholded Asymmetric Signal Perceptron

The ASP uses a single output species Y, therefore to translate a real-valued concentration as a Boolean, we compare the concentration of Y with the 0.5 threshold externally (by an outside observer). Now, since the input concentrations are fixed, but the output concentration is not, multiple perceptrons connected in a cascade may not work properly without extra precautions. We cannot therefore claim that either of our designs is modular. To address, that we introduce a tresholded version of the ASP, the TASP, which amplifies the output to the same level as expected by the input by employing Wilhelm's bistable chemical reactions as introduced in Section 3.3. We can easily adjust this mechanism for a custom upper value and threshold, and derive the TASP, which requires one extra species Y_{aux} and four reactions, as shown in Table 4.4(b). Recall that the bistable chemical system has three equilibrium states: lower value 0, threshold $(k_1 - \sqrt{k_1 D})/2k_3$, and upper value $(k_1 + \sqrt{k_1 D})/2k_3$, with discriminant $D = k_1 - 4k_3k_4$. If we set the upper value—the input concentration of the ASP—to 1.5 we calculate the rate constants as $k_1 = 1$, $k_2 = 1$, $k_3 = 0.533$, and $k_4 = 0.3$. Note that besides output Y, its companion species Y_{aux} is amplified as well (upper value 2.25). The threshold for the given rate constants is 0.375 for

Y and 0.141 for Y_{aux}.

Note that an active thresholding or a conditional amplification is an additional feature built on the top of the standard ASP (SASP). Other than the output interpretation, all features and settings, including the input-weight integration and learning, are the same. Therefore, the label ASP will refer to both models. If a distinction is needed, we will use either the standard ASP (SASP) or its thresholded extension (TASP). Unlike the SASP, in which the input species are the sole fuel, the TASP continuously consumes external species, which is kept constant to maintain the output at the upper equilibrium.

4.2.5 Execution

So far, we have introduced the WLP, the WRP and the ASP structurally as a collection of species, reactions, catalysts, and inhibitors that model the input-weight integration to mimic any linearly separable binary function. To illustrate this capability, we execute the chemical perceptrons with the best rate constants found by evolution.

Let us assume we know the correct weight species concentrations for a given binary function and, as a first step, we place the weight species molecules into the tank. (Note that normally we would obtain the weight species concentrations by learning.) Then, we inject one of the input combinations from Table 4.1. The concentration of each input species and the input signal is 1 for the WLP, 2 for the WRP, and 1.5 for the ASP. For instance, the input $(x_1, x_2) = (1, 0)$ is injected to the WLP as $[X_1^1] = 1.5$ and $[X_2^0] = 1.5$, as $[X_1^1] = 2$ and $[X_2^0] = 2$ to the WRP, and as $[S_{in}] = 1.5$ and $[X_1] = 1.5$ to the ASP.

Since this processing of inputs and producing the output takes some time, we cannot inject the input species immediately after the previous pair. We have to wait until the system settles down. For the WRP, the length of this period is $S_{WLP/WRP} = 5,000$ steps, for the ASP it is reduced to $S_{ASP} = 1,000$ steps.
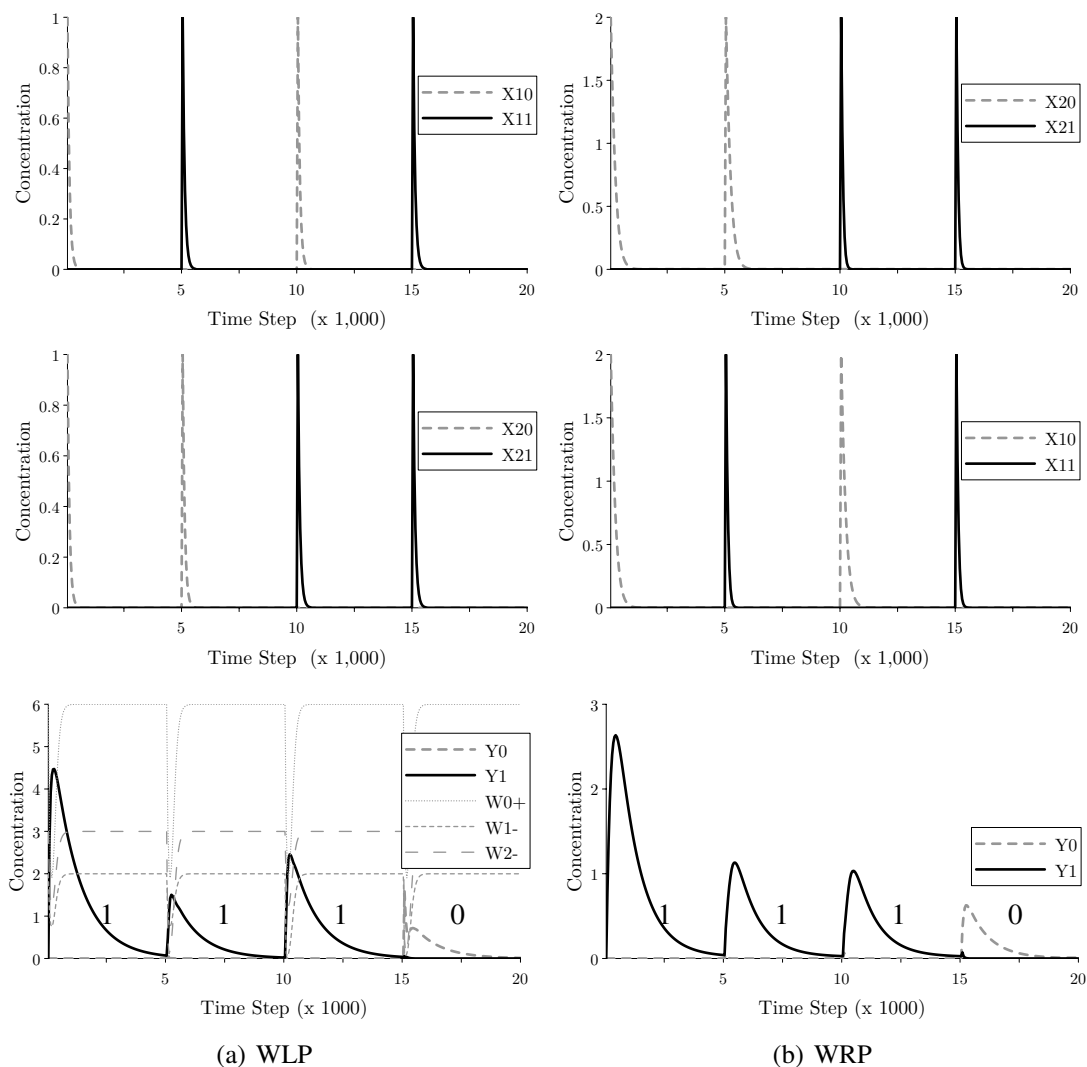
Figure 4.5: Simulation of the NAND function on four different combinations of the input species by (a) the WLP, (b) the WRP. From top to bottom: concentration of input species X_1^0, X_1^1, concentration of input species X_2^0, X_2^1, and concentration of output Y^0, Y^1. By applying the translation that compares the maximal concentration of Y^1 and Y^0 in the four intervals 5,000 steps long, we obtain the NAND output sequence $1, 1, 1,$ and 0.

(a) SASP

(b) TASP
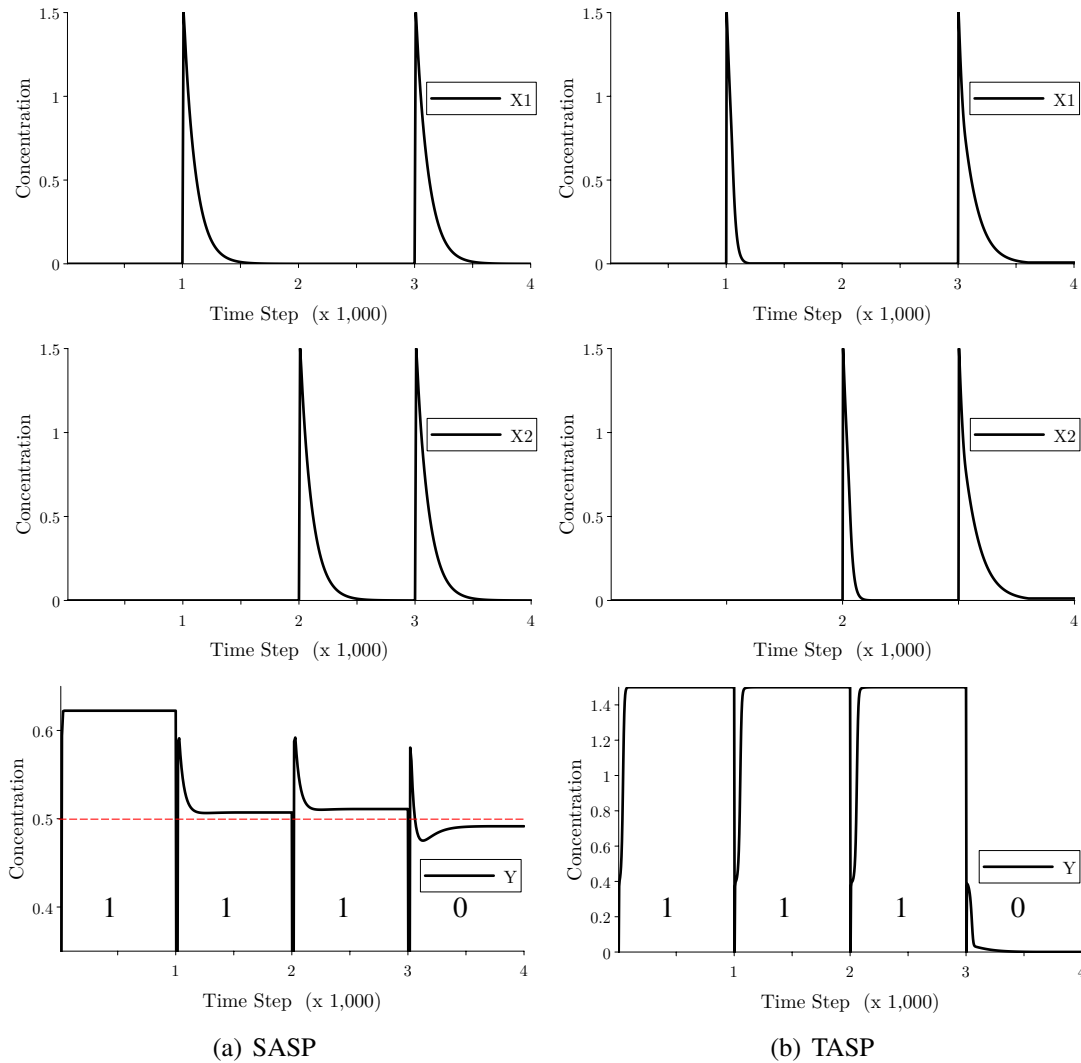
Figure 4.6: Simulation of the NAND function on four different combinations of the input species by (a) the SASP, and (b) the TASP. From top to bottom: concentration of input species X_1, concentration of input species X_2, and concentration of output Y. By applying the translation that (a) externally interprets threshold $[Y] > \Theta = 0.5$, and (b) distinguishes between $[Y] = 0$ and $[Y] = 1.5$ in the four intervals 1,000 steps long, we obtain the NAND output sequence $1, 1, 1,$ and 0.

Figures 4.5 and 4.6 present the WLP, WRP and ASP (both SASP and TASP) execution traces on four consecutive inputs with the concentration of weight species set according to the `NAND` function. Going from left to right, we obtain the `NAND` function outputs $1, 1, 1, 0$. The WLP and the WRP output one if, following the annihilation of output species, the species Y^1 remains (solid peaks), otherwise Y^0 (dashed peaks) indicates the binary output zero. For the SASP, the concentration of the single species Y in terms of its position above or below the threshold determines the output. This is what we call passive thresholding. On the other hand, the TASP actively distinguishes between these two positions and amplifies or diminishes the output accordingly. Unlike the WLP/WRP, the output species does not decay in the ASP, hence it must be discarded after each processing.

4.3 LEARNING AND FEEDBACK

The input-weight integration part deals with the output production driven by a given concentrations of the weights. To alter the predefined weight concentrations, and therefore to alter the predefined functionality, we train the perceptron such that it adheres to the required input-output profile. In this section we describe two approaches, learning by desired output and reinforcement learning, incarnated in our chemical designs.

The original definition of the formal perceptron learning (Section 2.1.1) adapts a weight w_i as $\Delta w_i = \alpha(d - y)x_i$ for a given output y and desired output d. Assuming $d \neq y$, each weight participating in the output production, i.e., $x_i = 1$, increments by $\Delta w = \alpha(d - y)$. Since the sign of the weight sum fully determines output, weight adaptation is stronger for inputs with the higher number of ones. For instance, the weight sum is adjusted by Δw for input $(0, 0)$, but $3\Delta w$ for input $(1, 1)$, as shown in Table 4.5(a). A learning rate α is incorporated into our chemical learning as the concentration of either

Table 4.5: The adaptation of a weight sum during a learning of two-input binary perceptron for four inputs: (a) a uniform adaptation of individual weights, (b) a uniform adaptation of weight sum.

	(a)			(b)	
x_1	x_2	Adapted Weight Sum	x_1	x_2	Adapted Weight Sum
0	0	$w_0 + \Delta w$	0	0	$w_0 + \Delta w$
1	0	$w_0 + w_1 + 2\Delta w$	1	0	$w_0 + w_1 + \Delta w$
0	1	$w_0 + w_2 + 2\Delta w$	0	1	$w_0 + w_2 + \Delta w$
1	1	$w_0 + w_1 + w_2 + 3\Delta w$	1	1	$w_0 + w_1 + w_2 + \Delta w$

the desired output D, or the penalty signal P. The chemical perceptrons do not age, i.e., a learning rate α is constant throughout the whole training.

Essentially, the uniform adaptation of individual weights causes a bias in the weight adaptation. Our simulations showed that this unfairness hurts the overall performance and is also biologically questionable, since the overall amount of Δw would need to reflect the number of inputs and the number of those holding logic one. To fix that, we do not adapt individual weights but the whole weight sum uniformly for all inputs as presented in Table 4.5(b). More specifically, the chemical perceptron divides Δw—the concentration of the desired output D or the penalty signal P—among weights, so the whole weight sum is adapted by Δw. We think that the biased adaptation in the original formal perceptron does not cause substantial issues because the weight sum is further processed by an activation function and the learning rate α decreases over time. As a result, small differences in the weight adaptation become neglectful.

4.3.1 Learning by Desired Output

The WLP and the WRP learn binary functions by supervised learning based on desired output. They share the same learning-embedded reactions, hence the description we provide in this section is valid for both.
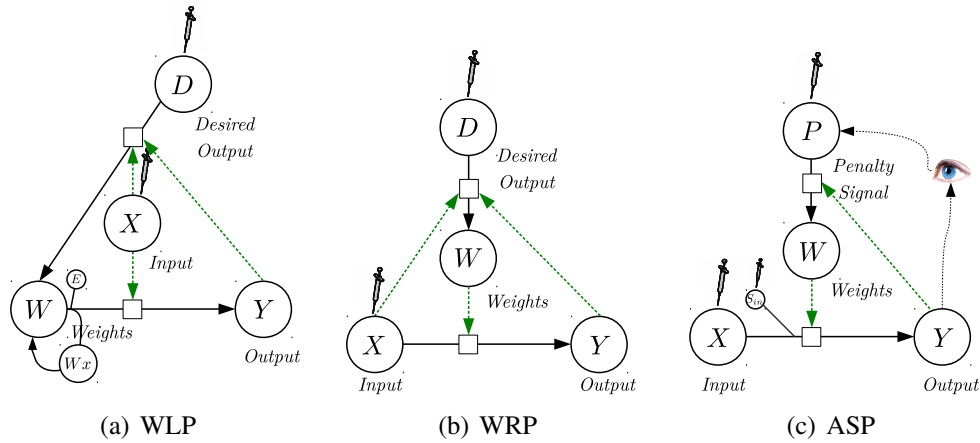
Figure 4.7: Overall qualitative diagram covering the reactions of the input-weight integration as well as learning for (a) WLP, (b) WRP, and (c) ASP.
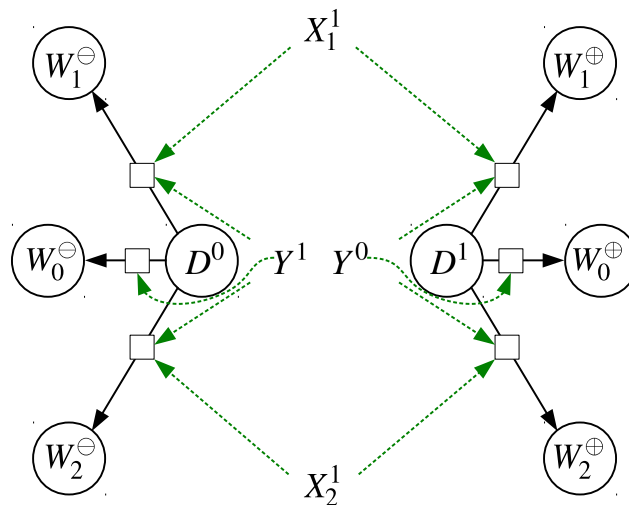
Figure 4.8: The WLP's (and WRP's) reactions employed in learning. The desired-output species D^0 or D^1 transforms to weights W, if the provided variant does not match the actual output, Y^0 or Y^1.

Recall from Section 2.1.1 that classical supervised learning expects a trainer to feed the perceptron with the binary desired output d. Figure 4.7(a) shows a high level diagram of the WLP covering both the input-weight integration as well as learning. Figure 4.7(b) shows the same for the WRP. By applying the representation symmetry, the desired output d translates into two species D^0 and D^1, and so during each learning step, the WLP/WRP

compares the variant of the actual output Y against the variant of the desired output D. If Y matches D, i.e., the species Y^0 and D^0, or Y^1 and D^1 are simultaneously present in the system, the output is correct, and the weights remain unaltered (D^0 or D^1 disappears by decay). Otherwise, the desired-output species D transforms to the $\oplus$ or $\ominus$ version of the weight species W, which are added to (or annihilate with) existing weights. This happens, however, only for those weights that participate in an output production for the current inputs (Figure 4.8). Thus, an input together with an output catalyzes the $D \rightarrow W$ reactions, so they are dependent (AND) catalysts (Table 4.3(a), Group 11 or Table 4.3(b), Group 9). Because the bias weight W_0 always participates in the output production it gets adapted no matter what input was injected. On the other hand, the weight species W_1 and W_2 race on their specific input substrate X_1^1 and X_2^1, respectively. The learning rate α is defined as the concentration of the desired output species, so the more D we provide, the more the weight concentrations change.

4.3.2 Learning by Penalty Signal

As opposed to the WLP or the WRP, the ASP needs just a single output species Y. Here we interpret the formal binary output by thresholding. As a matter of fact, following the same approach as before and distinguishing the desired output by two variants D^0 and D^1 would be cumbersome. Recall that WLP/WRP requires two species—the input and the actual output—to simultaneously catalyse the transformation of the desired output to the weights. This part of the design is rather artificial since most common reactions have a maximum of one catalyst. The ASP promised fewer and simpler reactions, therefore, we avoid using more than one catalyst simultaneously.

For the ASP we choose a more biologically plausible alternative for the supervised desired output learning—learning by reinforcement [147]. We introduce a special event

signal, the penalty signal P, which represents a reinforcement for an incorrect output (Figure 4.7(c)). Now, ASP has to decide whether to increase or decrease the concentrations of the weight species W_0, W_1, and W_2. We represent those two options by intermediate species $W^\oplus$ and $W^\ominus$, and so we let two reactions $P \rightarrow W^\oplus$ and $P \rightarrow W^\ominus$ compete on the penalty signal P. Following our asymmetric approach to comparison, only one of the reactions, namely $P \rightarrow W^\ominus$, has a catalyst Y. The concentration of Y decides whether weights will be incremented or decremented, and the concentration of P defines by how much (the learning rate α). More precisely, since the concentration $[Y] > 0.5$ represents one, the presence of the penalty signal P for a high concentration of Y means we expected ASP to provide zero, therefore the weight concentrations must drop, and so P should split to more $W^\ominus$ than $W^\oplus$ molecules. Note that as a consequence of the reinforcement both variants $W^\ominus$ and $W^\oplus$ are always produced. Also, compared with the WRP, the ASP does not have to handle the disappearance of the feedback species P because if the ASP operates as expected, we skip the injection of P.

Having $W^\oplus$ and $W^\ominus$, the second step is to decide which weights should be adapted. All reactions employed for learning are presented in Figure 4.9. Similarly to the WRP, only the weights responsible for the current output production are altered. Since the bias weight W_0 is active for the input (clock) signal S_{in}, which is always present, we can directly draw the reaction from $W^\oplus$ to W_0 and annihilate $W^\ominus$ and W_0 for the weight decrease. By using annihilation we avoided introducing the intermediate $W_0^\ominus$ species. Again, similarly to the WRP, the weights W_1 and W_2 are active only for the inputs X_1 and X_2, respectively. Thus, we need their substrates from the input-weight integration to catalyse the transformation to the intermediate weight changers $W_1^\ominus$ and $W_2^\ominus$ (which annihilate with weights W_1 and W_2), or the transformation of $W^\oplus$ directly to W_1 and W_2.

The ASP requires more reactions (10) than WLP/WRP (8) to model learning (Table

Figure 4.9: The ASP's reactions employed in learning. The penalty signal P increments or decrements the weights W depending on the concentration level (low vs high) of the actual output Y produced.

4.3 and 4.4) mainly because it forbids two simultaneous catalysts per reaction; otherwise, we would have achieved an even larger reduction in the number of reactions. Also, we intentionally do not handle the case where the concentrations of the weight-specific concentration changers $W_1^\ominus$ and $W_2^\ominus$ exceed the actual concentrations of the weights W_1 and W_2, respectively. We assume this situation does not occur thanks to a sufficiently high starting concentration of weights and low concentration of the penalty signal. We satisfy these properties by a genetic search of the rate constants (Section 4.4.1).

4.3.3 Execution

Here we present an experiment execution protocol of the chemical perceptrons to demonstrate their learning capabilities. Note that similarly to Section 4.3.3, the presented examples have the rate constants set by genetic algorithms.

The initial concentrations of the weight species are drawn from a uniform distribution

on the interval [2, 10] for the WLP and the WRP, and the interval [0.5, 1.5] for the ASP. The learning rate α is constant throughout the whole training, which translates into the constant concentration of feedback species—the desired output $[D^0] = 2$ or $[D^1] = 2$ for the WLP and the WRP, and the penalty signal $[P] = 0.2$ for the ASP. The feedback species together with the input species describe the expected input-output behaviour of the perceptrons. We determined the optimal concentration of feedback species by experiments. If the concentration is too high, the weights would oscillate and would not converge on a stable solution. Conversely, a low concentration of the feedback species (i.e., low learning rate) prolongs the learning process and does not provide enough pressure to drive weights out of the zero region if their concentrations are very low.

After injection of a single input, we need to allow the chemical perceptron some time to produce the output. If we injected inputs together with the feedback at the same time, the adaptation of the weights would start immediately, changing the actual output. Thus, the actual output would differ from the one we would obtain only by providing input species. In the extreme case, the WLP/WPR could just copy the desired output to the actual output by having very low concentrations of weights. To prevent this, we inject an input, wait a certain number of steps, measure the output, and then provide the feedback. Note that both input species X and output species Y must be present in the moment of weight adaptation. Therefore, we must allow enough time for the output production, but we cannot postpone the injection of the feedback species for too long, otherwise the chemical perceptron would fully process the input species. We found experimentally that this delay can be fairly short. More precisely, in our learning simulations we wait 100 simulation steps and then automatically inject the desired output (WLP/WRP), or first verify the correctness of the output, and then provide a penalty or no signal (ASP). This also means that the ASP, unlike the WLP and the WRP, requires active participation of

the trainer or environment. During each learning iteration we randomly draw one of the four input and feedback combinations and repeat this process every $S_{WLP/WRP} = 5,000$, or $S_{ASP} = 1,000$ steps until a solution is found.

Figures 4.10 and 4.11 present a trace of the WLP, the WRP, and the ASP (both SASP and TASP) execution for learning the NAND function, starting from a state where the weight concentrations are set such that they represent the FALSE function. Over several learning iterations the concentrations of the weight species change towards the expected solution.

(a) WLP   (b) WRP
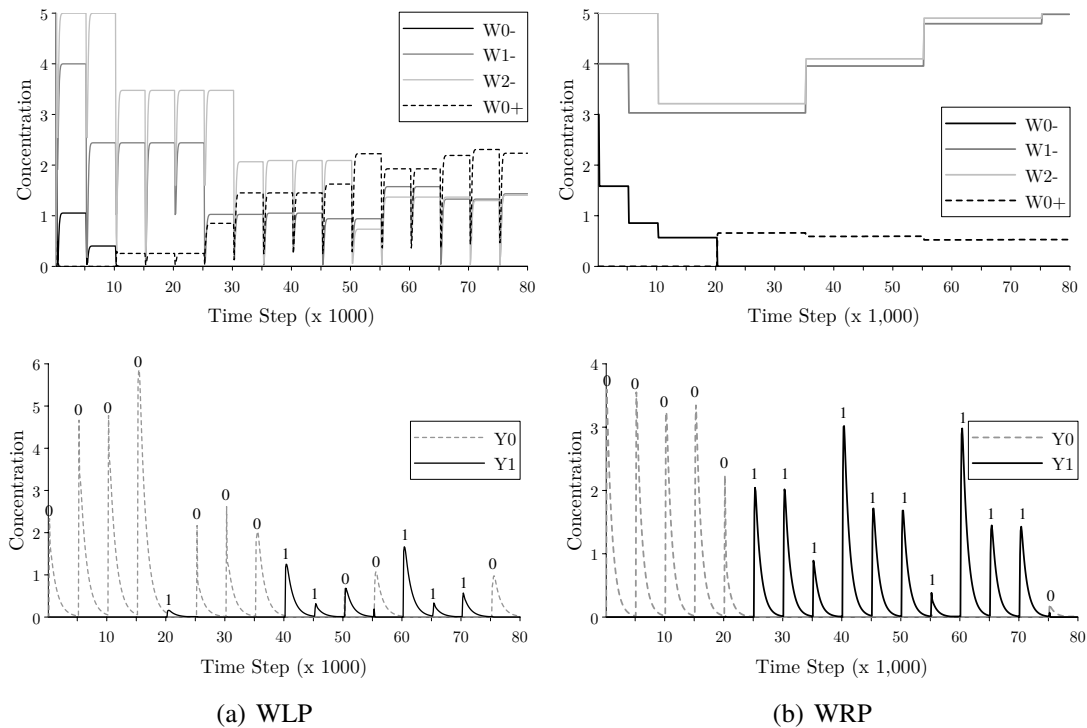
Figure 4.10: Training of (a) the WLP, and (b) the WRP to perform the NAND function starting from the FALSE setting: adaptation of weights (top), and output (bottom). Constant zeros gradually change to the NAND function outputs $1, 1, 1, 0$. Note that we have replaced a random order of training samples by a fixed order solely for illustrative purposes.

Figure 4.11: Training of (a) the SASP, and (c) the TASP to perform the `NAND` function starting from the `FALSE` setting: adaptation of weights (top), and output (bottom). Constant zeros gradually change to the `NAND` function outputs $1, 1, 1, 0$. Note that we have replaced a random order of training samples by a fixed order solely for illustrative purposes.

4.4 PERFORMANCE AND RESULTS

In this section we present the results of our simulations, covering the learning performance of the WLP, the WRP, the SASP variant with Michaelis-Menten (SASP MM) and mass-action kinetics (SASP MA), and the TASP with the same two kinetics—TASP MM and TASP MA. Unlike the WPR, the ASP's reactions do not contain two simultaneous catalysts, so we can directly rewrite catalytic reactions in the mass-action format using standard expansion. In doing so we prepare the ASP for a potential DNA-strand displacement implementation [128, 140, 166].

4.4.1 Genetic Search

Recall that the WLP, the WRP and the ASP were introduced as a collection of species and reactions, avoiding the specification of rate constants. Since the space of possible rate constants is large, it would be difficult and time-consuming to sample it in a trial-and-error fashion or by exhaustive search. We therefore employ a standard genetic algorithm (GA) [46, 123] to optimize the rate constants. Our reaction design is a qualitative model, which becomes a quantitative, ODE-driven system once the rate constants are set.

Chromosomes encoding possible solutions are simply vectors of rate constants, which undergo cross-over and mutation. The fitness of a chromosome reflects how well a chemical perceptron with the given rate constants (encoded in the chromosome) learns the given binary function. As mentioned in Section 4.3.3, during each learning iteration the chemical perceptron obtains one of the four input and feedback combinations. Each training consists of 120 learning iterations; however, we count only the last 20 iterations. The fitness of a single chromosome is then the average over 150 runs for each of the binary functions. We included the detailed GA parameter values in the Appendix, Table D.2.

For the WLP and the WRP the GA reaches solutions with the fitness above 0.9 already within a couple of generations, after which it continues with a slower pace toward the maximum fitness 1, which is reached around generation 20. Overall, the fitness landscape of the rate constants for the WLP and the WRP has the shape of a high table plateau, hence finding acceptable rate constants is not difficult. This demonstrates that their structural design in terms of species and reactions already provides a correct behavior, and the perceptrons do not need to rely much on the specific rate constants as shown in Section 4.4.3.

We performed 20 evolutionary runs for both SASP variants. In all cases, the fitness

quickly climbs above 0.7 and then either settles to a local optimum around $0.75 - 0.9$ or, in 25% of cases, it reaches the maximal values of $0.99 - 1.0$. Since the SASP's output Y is not produced steadily—it does not act as a constant influx—we could not apply the same rate constants and calculate just the thresholding reactions analytically, therefore, we had to run the GA for the TASP as well. Almost all TASP's evolutionary runs saturated at the maximal fitness, similarly to the WLP and the WRP.

4.4.2 Learning Performance

Since we are interested only in the best performing instances, we obtained learning performance for the best GA rate constants only (Appendix, A.3). We calculated the average learning success rate over $10,000$ simulation runs for each of 14 binary functions, where each run consists of 200 training iterations, similar to the fitness evaluation (see Section 4.4.1). The results (Figure 4.12) show that all chemical perceptrons can successfully learn 14 logic functions. The WLP and the WRP reach the perfect score of 100%, the ASP reaches a nearly perfect score of 99.5% (SASP MM), 99.3% (SASP MA), 99.999% (TASP MM), and 99.995% (TASP MA). That illustrates the asymmetric ASP design is correct and works properly, even with just a half of the WRP's number of reactions. Note that performance of a formal non-chemical perceptron with a signum activation function and a constant learning rate $\alpha = 0.1$ included in Figure 4.12 reaches 100% accuracy after 152th learning iteration, which is comparable with the WLP and the WRP chemical perceptrons.

The WLP and the WRP start from a balanced distribution where the probabilities of the $\oplus$ and the $\ominus$ weight species are equal, and hence initially the output production is evenly split between Y^0 and Y^1. Furthermore, because of the symmetric design, the learning difficulty of a function and its complement are the same (e.g., OR and NOR). Because
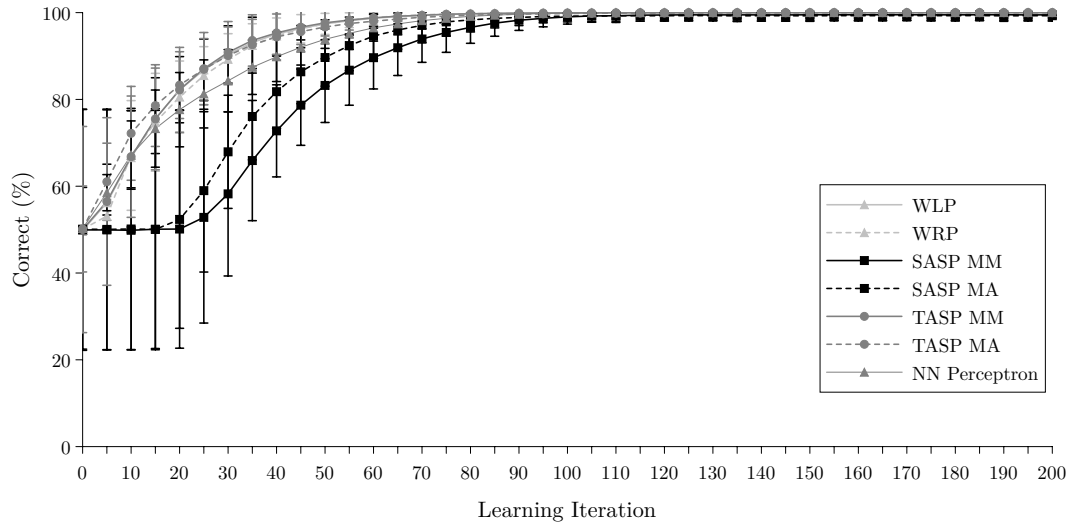
Figure 4.12: Mean and standard deviation of the 14 correct learning rate averages for the WLP, the WRP, the SASP MM, the SASP MA, the TASP MM, the TASP MA, and a formal perceptron with a signum activation function. Each average corresponds to one linearly separable binary function, for which 10,000 runs were performed.

of the asymmetry, it is challenging to find a balanced initial concentration range for the SASP. In fact, the GA in all our evolutionary runs drives the rates to the state where the initial probability of output one ($[Y] > 0.5$) and output zero ($[Y] \leq 0.5$) differs. For the best rate constants the SASP always starts with a zero output as a `FALSE` function (Figure 4.13). Therefore, the SASP has a bias for functions with more zeros in the output, and the learning difficulty of a binary function and its complement do not match in general.

Further, the SASP's error is very function-specific. For instance, the `NAND` function has by far the worst performance, i.e., 94.78% (SASP MM), 96.02% (SASP MA). Since the SASP acts initially as `FALSE`, it must push the output for all but the last bit above the threshold (Figure 4.11). `NAND` is also the most difficult function because it is non-additive and only the simultaneous presence of X_1 and X_2 with a low concentration of W_1 and W_2 annihilates the output below the threshold. Besides `NAND`, only the functions `AND`, `IMPL`, and `CIMPL` reach non-perfect scores of $97.5 - 99.5\%$. Even though the error is marginal,
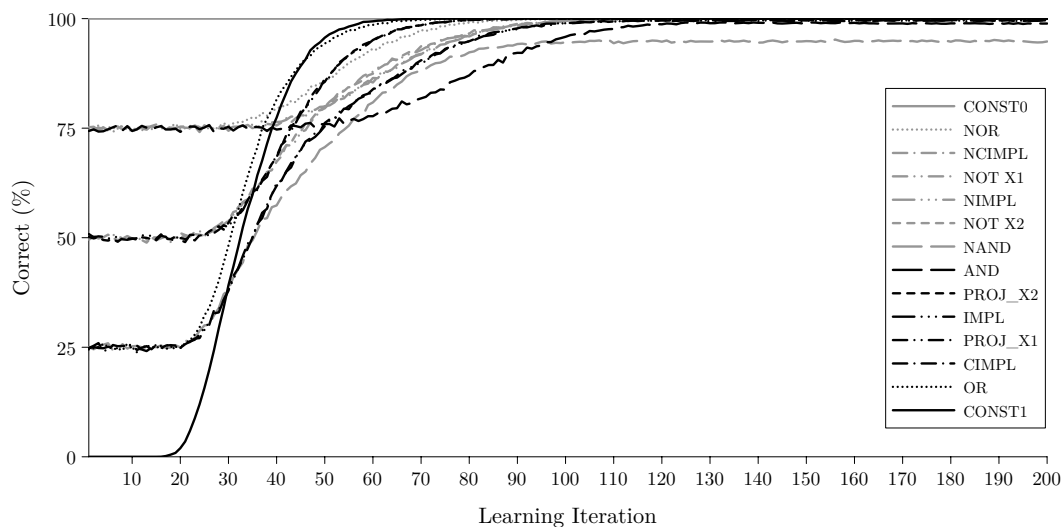
Figure 4.13: Performance of the SASP MM for each of the 14 linearly separable binary functions averaged over 10,000 runs runs. The performance of the mass-action variant is similar and not shown here.

we speculate it could be eliminated by conducting a more detailed search on the initial weight and input concentrations.

On the other hand, the TASP's error is negligible ($< 0.005\%$) and it reaches the expected behavior faster than the SASP. This is due to a larger gap between the formal output one (1.5) and zero (0), which makes the net effect of the weight-changer reactions responsible for the reinforcement learning driven by the output species qualitatively more distinct. The TASP, compared with the SASP, starts with less unbalanced weight concentrations but still its initial preference of the output zero is 94% for the TASP MM. The TASP MA prefers the output one at 64%. This also implies that the space of possible solutions with different starting bias for the TASP is larger than for the SASP. Table 4.6 summarizes the features of all our chemical perceptrons.

Table 4.6: Comparison of the binary chemical perceptrons: the WLP, the WRP, the SASP, and the TASP. The learning performance of all models is almost equivalent and reaches ~ 100% accuracy. The WLP and the WRP, which employ a symmetric design, are substantially larger then the asymmetric binary chemical perceptrons of the SASP and the TASP. On the other hand, the asymmetric perceptrons are less robust to rate constant perturbation.

Attribute	WLP	WRP	SASP	TASP
Number of species	21	14	12	13
Number of reactions	34(54)	30	16	20
Overall learning performance	100%	100%	99.5% (MM)	99.999% (MM)
			99.3% (MA)	99.995% (MA)
Rate robustness (50% perturb.)	98.98%	99.34%	86.8% (MM)	63.65% (MM)
			85.62% (MA)	65.76% (MA)
Symmetry of species	yes	yes	no	no
Output interpretation	$[Y^1] > [Y^0]$	$[Y^1] > [Y^0]$	$[Y] > .5$	$[Y] = 0, [Y] = 1.5$
Learning feedback	desired output	desired output	penalty signal	penalty signal
Time steps per input processing	$5,000$	$5,000$	$1,000$	$1,000$
ODE numerical approximation	Euler	Euler	RK 4	RK 4

4.4.3 Robustness Analysis

We have shown that the chemical perceptron performs very well by learning all functions almost perfectly (~ 100% accuracy). The question arises how much the performance depends on the rate constants, and whether we can tolerate errors in an implementation? To answer that question, we introduce perturbations and study the system under different conditions. We replace each rate constant γ randomly by $(1 \pm q)\gamma$, where q is drawn from a uniform distribution over the interval $(0, p)$, where p is the perturbation magnitude (strength).

We analyzed the robustness of the chemical perceptrons (Figure 4.14) with the best rate constants only. The results show that the WLP and the WRP maintain very high robustness, even for 50% perturbation, its learning rate approaches 98.98% (WLP) and 99.34% (WRP). The main difference between the WLP and the WRP occurs at high perturbation when the WLP becomes slightly more vulnerable. For the 200% perturbation magnitude, the performance of the WLP drops even below 50%, which is a sign that the

concentration of output species rises beyond the maximal limit in some simulations. Note that for the WLP, this situation might happen due to an open fuel influx.

On the other hand, the SASP is also fairly robust—for a 50% perturbation its learning rate reaches 86.8% (MM) or 85.62% (MA), although compared with the WLP or the WRP, the SASP's robustness is significantly lower. Beyond the 125% perturbation magnitude, the gap between the WLP/WRP and the SASP starts to shrink, and finally the performance reaches $55 - 60\%$ for the 200% perturbation, which basically means that the rate constants are selected at random. The SASP is less robust because of the asymmetric real number representation, where a single catalyst represents both a positive and negative number, depending on its concentration and rate. Therefore, by perturbing a single reaction rate we might alter the functioning of the system more dramatically than for the symmetric representation. In the WLP/WRP, real numbers are expressed structurally by having two mirrored catalytic reactions racing on the same substrate. The design is thus more redundant and robust. In the opposite case, since the SASP requires a smaller number of reactions and rates, it relies more on its individual components and is therefore less robust.

For the TASP a perturbation would produce a system that is still bistable, i.e., distinguishes between the logic one and zero values separated by the threshold, but these values would most likely differ from than the original ones, which we calculated analytically to match the expected output of 1.5 (logic one). That is why we relaxed the output interpretation and set it to the SASP's $[Y] > 0.5$. Because the thresholding reactions are very sensitive to any rate change, the robustness of the TASP, even with the relaxed interpretation is quite small—just 63.65% (MM) or 65.76% (MA) for a 50% perturbation.
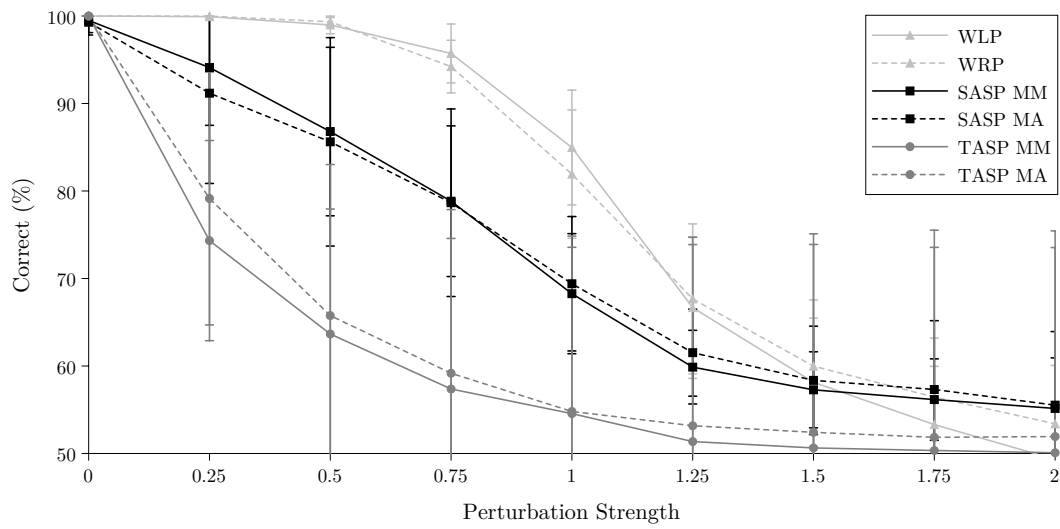
Figure 4.14: Mean and standard deviation of the 14 final correct rate averages under the perturbation of rate constants after 200 learning iterations for for the WLP, the WRP, the SASP MM, the SASP MA, the TASP MM, and the TASP MA. Each average corresponds to one linearly-separable binary function for which 10^4 runs were performed. For a given perturbation strength p, each rate constant is perturbed randomly over the interval $(0, p)$.

5

ANALOG CHEMICAL PERCEPTRON

In the previous chapter, we presented several types of binary chemical perceptron as the first proof-of-concept artificial chemical systems that can learn and adapt autonomously to the feedback provided externally by a teacher. The most advanced *asymmetric signal perceptron* (ASP) [22] requires less than a half of the reactions of its predecessors with comparable performance (i.e., $99.3 - 99.99\%$ success rates). The flip side of the more compact design is a reduced robustness to rate constant perturbations due to a lack of structural redundancy.

In real biomedical applications one is often required to distinguish subtle changes in concentrations with complex linear or nonlinear relations among species. Such behavior cannot easily be achieved with our previous binary perceptron models, thus, several improvements are necessary. In this chapter we present a new *analog asymmetric signal perceptron* (AASP) with two inputs. To avoid confusion, we will refer to the original ASP as a binary ASP (BASP). As usual, the AASP model follows mass-action and Michaelis-Menten kinetics and learns through feedback from the environment. The design is modular and extensible to any number of inputs. We demonstrate that the AASP can learn various linear and nonlinear functions. For example, it is possible to learn to produce the average of two analog values. This work has been published in parts in [20].

5.1 MODEL

The AASP models a formal analog perceptron [132] with two inputs x_1 and x_2, similar to an early type of artificial neuron [62]. It is capable of simple learning and can be used as a building block of a feed-forward neural networks. Networks built from perceptrons have been shown to be universal approximators [72].

While the previous BASP models a perceptron with two inputs and a binary output produced by external or internal thresholding, the new AASP is analog and does not use thresholding. Instead of a binary yes/no answer, its output is analog, which requires much

Table 5.1: (a) The AASP's species divided into groups according to their purpose and functional characteristics; (b) the AASP's reactions with the best rate constants found by the GA (see Section 4.4.1), rounded to four decimals. Reaction groups 1–4 implement the input-weight integrations, the rest implement learning. The catalytic reactions have two rates: k_{cat} and K_m.

Function	Species
Inputs	X_1, X_2
Output	Y
Weights	W_0, W_1, W_2
Target output	$\hat{Y}$
Input (clock) signal	S_{in}
Learning signal	S_L
Input contributions	X_1^L, X_2^L, S_{in}^L
Weight changers	$W^\ominus, W^\oplus,$
	$W_0^\ominus, W_1^\ominus, W_2^\ominus$
Total	17

#	Reaction	Catalyst	Rates
1	$S_{in} + Y \to \lambda$		.1800
2	$S_{in} \to Y + S_{in}^L$	W_0	.5521, 2.5336
3	$X_1 + Y \to \lambda$		.3905
	$X_2 + Y \to \lambda$		
4	$X_1 \to Y + X_1^L$	W_1	.4358, 0.1227
	$X_2 \to Y + X_2^L$	W_2	
5	$\hat{Y} \to W^\oplus$		.1884
6	$Y \to W^\ominus$	S_L	.1155, 1.9613
7	$Y + \hat{Y} \to \lambda$		1.0000
8	$W^\ominus \to W_0^\ominus$	S_{in}^L	0.600, 1.6697
9	$W_0 + W_0^\ominus \to \lambda$		.2642
10	$W^\oplus \to W_0$	S_{in}^L	.5023, 2.9078
11	$W^\ominus \to W_1^\ominus$	X_1^L	.1889, 1.6788
	$W^\ominus \to W_2^\ominus$	X_2^L	
12	$W_1 + W_1^\ominus \to \lambda$		.2416
	$W_2 + W_2^\ominus \to \lambda$		
13	$W^\oplus \to W_1$	X_1^L	.2744, 5.0000
	$W^\oplus \to W_2$	X_2^L	
Total	18		

finer control over the weight convergence. As a consequence, the AASP consists of more species, namely 17 vs. 13, and more reactions, namely 18 vs. 16.

5.1.1 Input-Weight Integration

A formal perceptron integrates the inputs $\mathbf{x}$ with the weights $\mathbf{w}$ linearly as $\Sigma_{i=0}^{n} w_i x_i$, where the weight w_0, a bias, always contributes to an output because its associated input $x_0 = 1$. An activation function φ, such as a hyperbolic tangent or signum, then transforms the dot product to the output y.

The reactions carrying out the chemical input-weight integration are structurally the same as in the BASP. The only difference is an addition of the partial input-weight contribution species, which are, however, required for learning only, and will be explained in Section 5.1.2. The AASP models a two-input perceptron where the output calculation is reduced to $y = \varphi(w_0 + w_1 x_1 + x_2 w_2)$. The concentration of input species X_1 and X_2 corresponds to the formal inputs x_1 and x_2, and the species Y to the output y. A clock (input) signal S_{in} is always provided along the regular input X_1 and X_2 because it serves as the constant-one coefficient (or the constant input $x_0 = 1$) of the bias weight w_0.

The AASP represents the weights by three species W_1, W_2, and W_0. As opposed to the formal model, the input-weight integration is nonlinear and based on an annihilatory version of the asymmetric representation of the values and the addition/subtraction operation as utilized in the design of the BASP and discussed generally in Chapter 3. Recall that since the concentration cannot be negative, we cannot map a signed real variable directly to the concentration of a single species. The weights require both positive and negative values, otherwise we could cover only functions that are strictly additive.

Using the asymmetric comparison primitives, we map the AASP's weights to catalysts, the inputs to substrates, and the output to product and obtain 6 reactions as shown in

Figure 5.1(a) and Table 5.1(b), groups 1–4. Each weight species races with its substrate's annihilation but also with other weights. Since the output Y is shared, this effectively implements a nonlinear input-weight integration. Note that by replacing annihilation with a decay of the input species, we would end up having three independent races with additive contributions instead of one global race. An alternative symmetric representation embedded in the previously reported *weight-loop perceptron* and the *weight-race perceptron* [21] encodes the values by two complementary species, one for the positive and one for the negative domain. We opt for the asymmetric approach because it reduces the number of reactions by half compared to the symmetric one.

Using Michaelis-Menten kinetics, the concentration ODEs for the input-weight integration reactions are

$$
\begin{aligned}
[\dot{X_i}] &= -\frac{k_{cat,i}[W_i][X_i]}{K_{m,i} + [X_i]} - k_i[X_i][Y] \\
[\dot{Y}] &= \sum_i \left(\frac{k_{cat,i}[W_i][X_i]}{K_{m,i} + [X_i]} - k_i[X_i][Y] \right),
\end{aligned}
\tag{5.1}
$$

where for consistency S_{in} is labelled as X_0.

Because of the complexity of the underlying ODEs, no closed formula for the output concentration exists and theoretical conclusions are very limited. Although we cannot analyze the input-weight integration dynamics quantitatively, we can still describe the qualitative behavior and constraints. The weight concentration represents formally both positive and negative values, so the weights together with annihilatory reactions can act as both catalysts and inhibitors. More specifically, a low weight concentration, which strengthens its input-specific annihilation, could impose a negative pressure on a different weight branch. Hence, we interpret a weight that contributes to the output less than its input consumes as negative. In an extreme case, when the weight concentration is zero,

its branch would consume the same amount of output as its input injected. The relation between the concentration of weights and the final output $[Y]_\infty$ has a sigmoidal shape with the limit $[X_1]_0 + [X_2]_0 + [S_{in}]_0$ reaching for all weights $[W_i] \to \infty$. Clearly the output concentration cannot exceed all the inputs provided.

Figure 5.2 shows the relation between the concentration of weight W_1 and weight W_2 and the final output concentration. For simplicity, the bias processing part is not considered ($[S_{in}] = 0$), so we keep only two branches of the input-weight integration triangle. Note that in the plots the concentration of weights span the interval 0 to 2 because in our simulations we draw the weights uniformly from the interval $(0.5, 1.5)$. On the z-axis we plotted the ratio of the output concentration $[Y]$ to $[X_1]_0 + [X_2]_0$. For learning to work, we want the gradient of the output surface to be responsive to changes in the weight concentrations. Therefore, we restrict the range of possible outputs so it is neither too close to the maximal output, where the surface is effectively constant, nor too close to zero, where the surface is too steep and even a very small perturbation of the weight concentration would dramatically change the output. Note that we optimized the AASP's rate constants to obtain an optimal weight-output surface by genetic algorithms (discussed in Section 5.1.3).

5.1.2 Learning

In the previous BASP model, learning reinforced the adaptation of weights by a penalty signal whose presence indicated that the output was incorrect. Since the output is analog in the new AASP model, a simple penalty signal is not sufficient anymore. We therefore replaced the reinforcement learning by classical supervised learning [130]. Formally, the adaptation of a weight w_i for the training sample $(\mathbf{x}, \hat{y})$, where $\hat{y}$ is a target output and $\mathbf{x}$ a input vector, is defined as $\Delta w_i = \alpha(\hat{y} - y)x_i$, where $\alpha \in (0, 1]$ is the learning rate. The

(a) Input-weight integration

(b) Output comparison

(c) Positive adaptation

(d) Negative adaptation
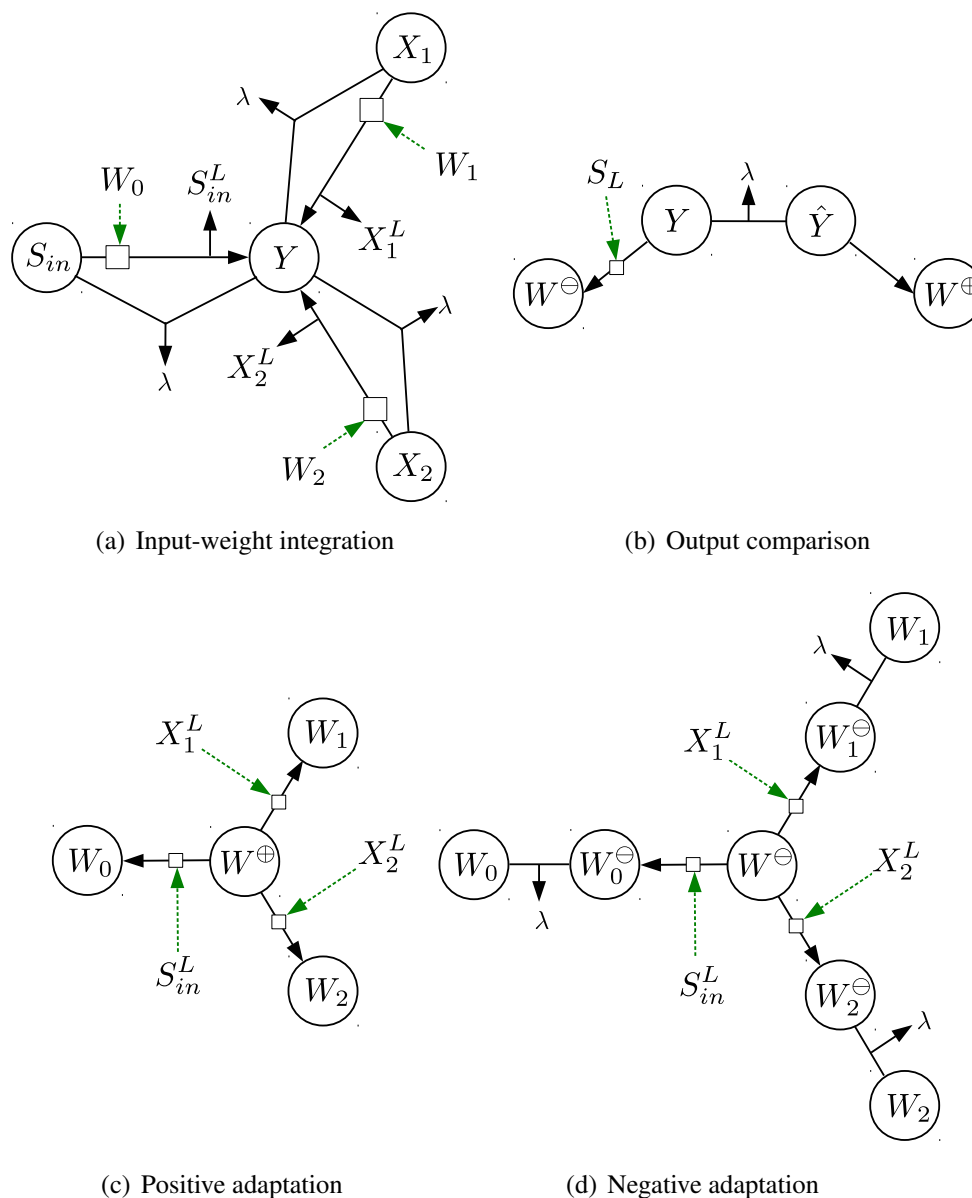
Figure 5.1: (a) The AASP's reactions performing input-weight integration. Similarly to the BASP, cross-weight competition is achieved by the annihilation of the inputs S_{in}, X_1, X_2 with the output Y, an asymmetric strategy for representation of real values and subtraction. (b-d) the AASP's reactions responsible for learning. They are decomposed into three parts: (b) comparison of the output Y with the target-output $\hat{Y}$, determining whether weights should be incremented ($W^{\oplus}$ species) or decremented ($W^{\ominus}$ species), and (c-d) positive and negative adaptation of the weights W_0, W_1, and W_2, which is proportional to the part of the output they produced S_{in}^L, X_1^L, and X_2^L respectively. Nodes represent species, solid lines are reactions, dashed lines are catalysts, and λ stands for no or inert species.
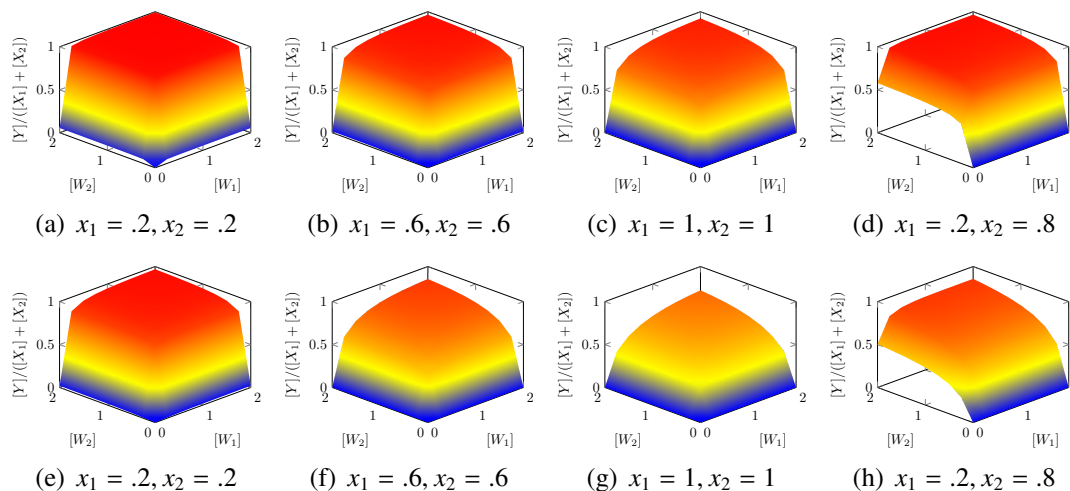
Figure 5.2: The relation between the weight concentrations $[W_1]$ and $[W_2]$ and the final output concentration $[Y]_\infty$ normalized by $[X_1]_0 + [X_2]_0$ for the input-weight integration (excluding the bias W_0 part) showing various inputs. The rate constant of annihilatory reactions $X_i + Y \rightarrow \lambda, i \in \{1, 2\}$ is $k = 0.2$ in the top and $k = 1$ in the bottom row.

AASP's, similarly to the BASP's input-weight integration, does not implement the formal Δw_i adaptation precisely, rather, it follows the relation qualitatively.

Learning is triggered by an injection of the target output $\hat{Y}$ provided some time after the injection of the input species. The part presented in Figure 5.1(b) compares the output Y and the target output $\hat{Y}$ by annihilation. Intuitively a leftover of the regular output Y implies that the next time the AASP faces the same input, it must produce less output, and therefore it needs to decrease the weights by producing a negative weight changer $W^\ominus$ from Y. In the opposite case, the AASP needs to increase the weights, hence $\hat{Y}$ transforms to a positive weight changer $W^\oplus$. Since the AASP can produce output also without learning, just by the input-weight integration, we need to guard the reaction $Y \rightarrow W^\ominus$ by a learning signal S_L, which is injected with the target output and removed afterwards. To prevent creation of erroneous or premature weight changers, the annihilation $Y + \hat{Y} \rightarrow \lambda$ must be very rapid. Note that the difference between the actual output Y and the desired

output $\hat{Y}$, materializing in the total concentration of weight changers $W^{\oplus}$ and $W^{\ominus}$, must not be greater that the required weight adaptation, otherwise the weights would diverge. The learning rate α is therefore effectively incorporated in the concentration of $W^{\oplus}$ and $W^{\ominus}$.

In the formal perceptron, the adaptation of a weight w_i is proportional to the current input x_i. Originally, the BASP distinguished which weights to adapt by a residual concentration of inputs X_1 and X_2. Because the inputs as well as an adaptation decision were binary, we cared only about whether some of the unprocessed input were still left, but not about its precise concentration. Thus, an injection of the penalty signal could not happen too soon, neither too late. Because the AASP's learning needs more information, the input-weight integration introduced three additional species, namely the partial input-weight contributions X_1^L, X_2^L, and S_{in}^L, which are produced alongside the regular output Y. A decision on which weights to update based on the input-weight contributions could be made even after the input-weight integration is finished. That allows to postpone an injection of the target output $\hat{Y}$ and the learning signal S_L.

Let us now discuss a positive adaptation as shown in Figure 5.1(c), where the total amount of $W^{\oplus}$ is distributed among participating weights. The input contribution species X_1^L, X_2^L, and S_{in}^L race over the substrate $W^{\oplus}$ by catalyzing the reactions $W^{\oplus} \rightarrow W_i, i \in \{0, 1, 2\}$. Note that the traditional weight adaptation formula takes into account solely the input value, so here we depart further from the formal perceptron and have the combination of input and weights compete over $W^{\oplus}$. Since larger weights produce more output they get adapted more. In addition, once a weight reaches zero, it will not be recoverable.

The negative adaptation presented in Figure 5.1(d) is analogous to the positive one, but this time the input-weight contributions race over $W^{\ominus}$ and produce intermediates $W_0^{\ominus}, W_1^{\ominus}$, and $W_2^{\ominus}$, which annihilate with the weights. Again, because the magnitude of a weight

update depends on the weight itself, this feedback loop protects the weight from falling too low and reaching zero, a point of no return. This is beneficial because, as opposed to the formal perceptron, a weight value (concentration) cannot be physically negative.

To implement the entire learning algorithm, the AASP requires 12 reactions as presented in Table 5.1(b), groups $5 - 13$.

5.1.3 Genetic Search

Since a manual trial-and-error setting of the rate constants would be very time-consuming, we optimize the rate constants by standard genetic algorithms (GA). Possible solutions are encoded on chromosomes as vectors of rate constants, which undergo cross-over and mutation. We use elite selection with elite size 20, 100 chromosomes per generation, shuffle cross-over, per-bit mutation, and a generation limit of 50. The fitness of a chromosome, defined as the RNMSE, reflects how well the AASP with the given rate constants (encoded in the chromosome) learns the target functions $k_1 x_1 + k_2 x_2 + k_0$, $k_1 x_1$, and $k_2 x_2$. The fitness of a single chromosome is then calculated as the average over 300 runs for each function. We included the $k_1 x_1$ and $k_2 x_2$ tasks to force the AASP to utilize and distinguish both inputs x_1 and x_2. Otherwise the GA would have a higher tendency to opt for a greedy statistical approach, where only the weight W_0 (mean) might be utilized.

Table 5.2: Target functions with constants k_1, k_2, k_0 drawn uniformly from the provided intervals, and mean and variance rounded to four decimal places.

$\hat{y}$	k_1	k_2	k_0	Mean	Variance
$k_1 x_1 + k_2 x_2 + k_0$	$(0.2, 0.8)$	$(0.2, 0.8)$	$(0.1, 0.4)$	0.85	0.0590
$k_1 x_1 - k_2 x_2 + k_0$	$(0.2, 0.8)$	$(0.0, 0.3)$	$(0.3, 0.4)$	0.56	0.0309
$k_1 x_1$	$(0.2, 0.8)$	–	–	0.30	0.0257
$k_2 x_2$	–	$(0.2, 0.8)$	–	0.30	0.0257
$k_1 x_1 x_2 + k_0$	$(0.2, 0.8)$	–	0.25	0.43	0.0154
k_0	–	–	$(0.1, 0.4)$	0.25	0.0075

5.2 PERFORMANCE

We demonstrate the learning capabilities of the AASP on 6 linear and nonlinear target functions as shown in Table 5.2. Simulations of the AASP were carried out by a 4th order Runge-Kutta numerical integration with the temporal step of 0.05. During each learning iteration we inject inputs X_1 and X_2 with concentrations drawn from the interval $(0.2, 1)$ and set the bias input S_{in} concentration to 0.5. We chose the target functions carefully, such that the output concentration is always in a safe region, which is far from the minimal (zero) and the maximal output concentration $[S_{in}]_0 + [X_1]_0 + [X_2]_0$. We then inject the target output $\hat{Y}$ with the learning signal S_L 50 steps after the input, which is sufficient to allow the input-weight integration to proceed.

For each function family we calculated the AASP's performance over 10,000 simulation runs, where each run consisted of 800 training iterations. We define performance as the *root normalized mean square error* (RNMSE)

$$\text{RNMSE} = \sqrt{\frac{\langle (y - \hat{y})^2 \rangle}{\sigma_{\hat{y}}^2}}, \qquad (5.2)$$

where the square error is normalized by $\sigma_{\hat{y}}^2$, a variance of the target output $\hat{y}$. A RNMSE of 1 means chance level. We also provide the results using another standard error measure: the *symmetric absolute mean percentage error* (SAMP) with values ranging from 0% to 100%

$$\text{SAMP} = 100 \langle \frac{|y - \hat{y}|}{y + \hat{y}} \rangle. \qquad (5.3)$$

The AASP's final RNMSE settles down to the range $(0.103, 0.0.378)$ (see Figure 5.4 and 5.3). When we include only the functions that utilize both inputs x_1 and x_2 as well

as the bias, i.e., the scenario the AASP was primarily designed for, the RNMSE drops to the range $(0.103, 0.304)$. Note that we do not distinguish between the training and testing set. During each iteration we draw the inputs with the target output for a given function independently.

Among all the functions, $k_1x_1 + k_2x_2 + k$ is the easiest (RNMSE of 0.103) and the constant function k_0 the most difficult one (RNMSE of 0.378). The function k_0 is even more difficult than the nonlinear function $k_1x_1x_2 + k_0$ (RNMSE of 0.304). Compared to the formal perceptron, the constant function does not reach a close-to-zero RNMSE because the AASP cannot fully eliminate the contribution (or consumption) of the X_1 and X_2 input-weight branches. The formal perceptron could simply discard both inputs and adjust only the bias weight, however, the AASP's weights W_1 and W_2 with zero concentration would effectively act as inhibitors, thus consuming a part of the output produced by the bias. Note that for the nonlinear function we set $k_0 = 0.25$, which does not increase the variance, i.e., only the nonlinear part counts toward the error. Figure 5.5 shows the weight concentration traces as well as the output, the target output, and the absolute error for selected functions.

(a) RNMSE

(b) SAMP
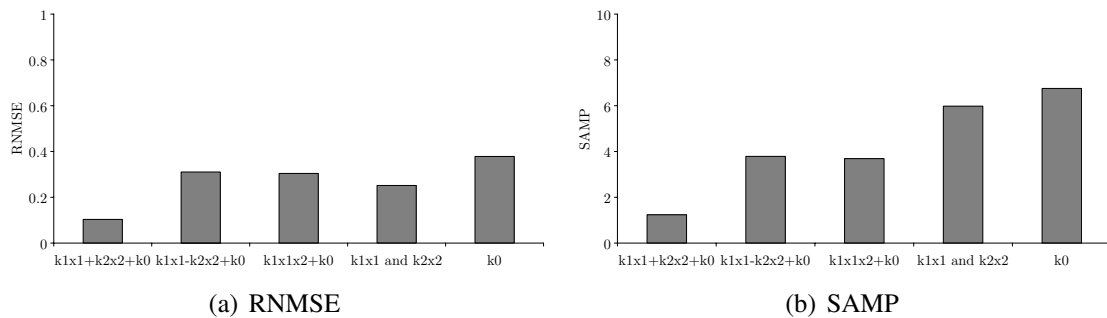
Figure 5.3: Final performance of the AASP on 6 linear and nonlinear functions after 800 learning iterations showing the error calculated as RNMSE and SAMP. Note that the final error for the functions k_1x_1 and k_2x_2 was taken at the 700th iteration because of a divergence that happen afterwards.
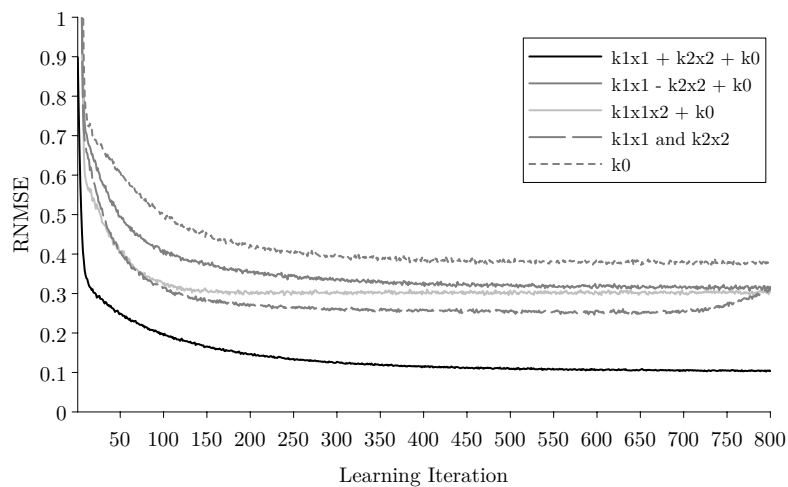
(a) RNMSE
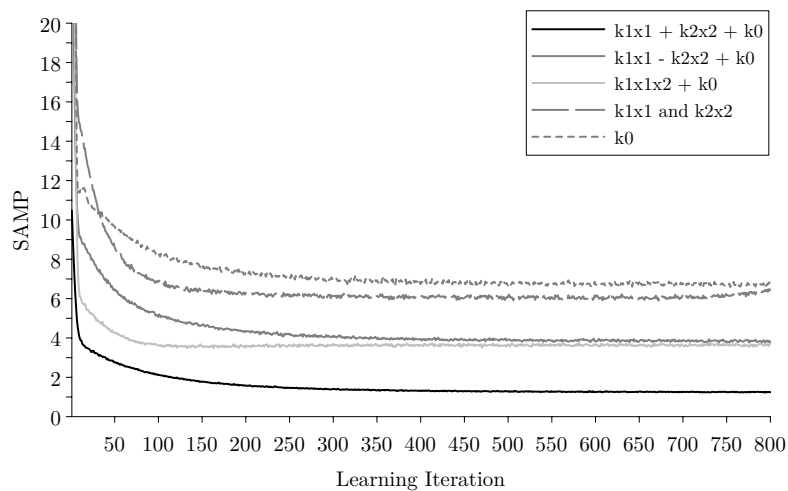
(b) SAMP

Figure 5.4: RNMSE and SAMP of the AASP on 6 linear and nonlinear functions over 800 learning iterations.

5.3 DISCUSSION

In this chapter we extended the chemical asymmetric design introduced for the asymmetric signal perceptron to an analog scenario. We demonstrated that our new AASP model can successfully learn several linear and nonlinear two-input functions. The AASP fol-

(a) $\hat{y} = k_1 x_1 + k_2 x_2 + k_0$

(b) $\hat{y} = k_0$

(c) $\hat{y} = k_1 x_1 x_2 + k_0$
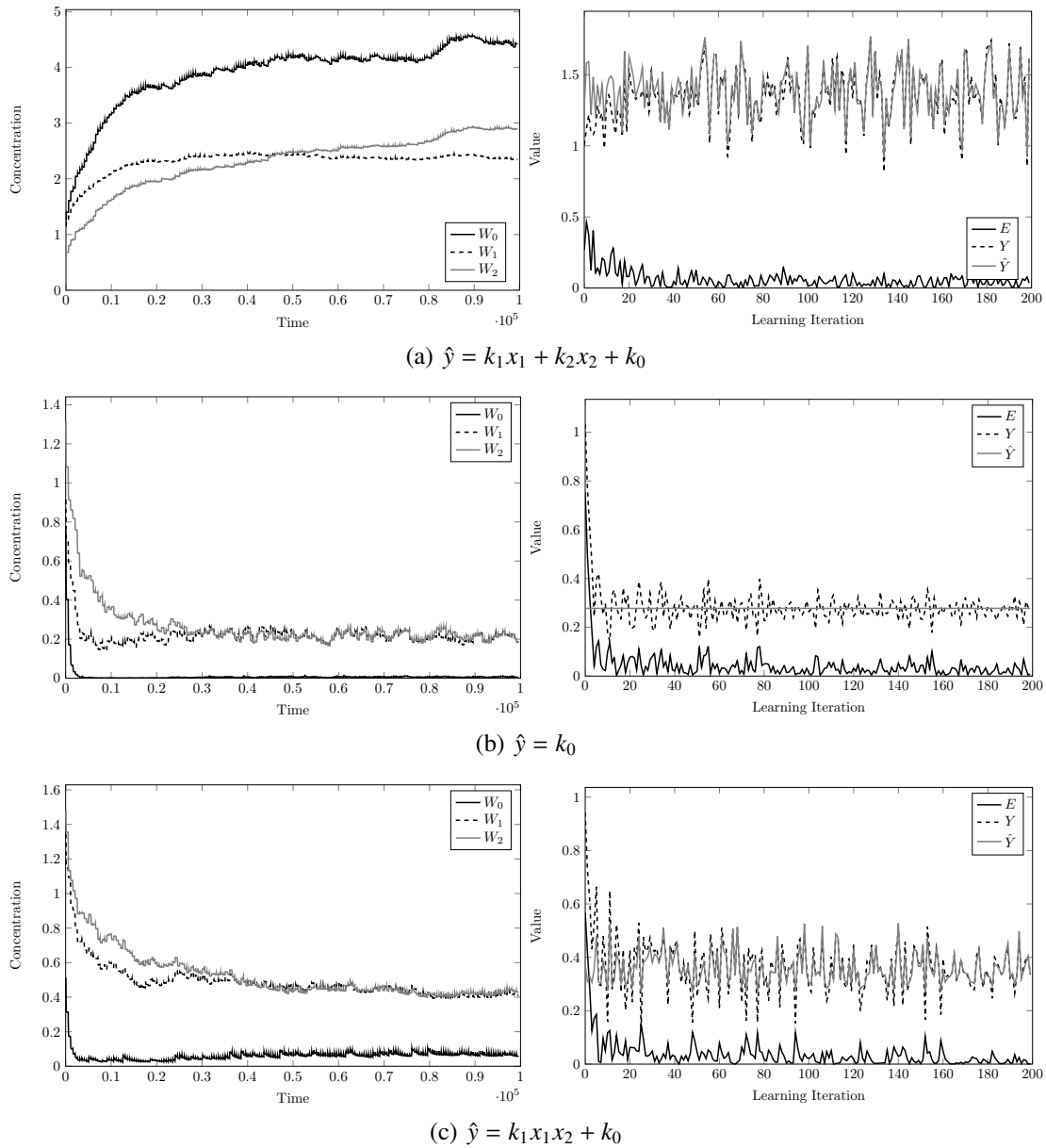
Figure 5.5: AASP learning examples for selected functions. The left column shows concentration traces of the weights, the right column the filtered output Y, the target output $\hat{Y}$, and the absolute error E.

lows Michaelis-Menten and mass-action kinetics, and learns through feedback provided as a desired output. As opposed to our previous designs that used simple binary signals, the AASP allows to adapt to precise concentration levels. In Chapter 6 we integrate the

AASP with a chemical delay line to tackle time-series prediction rather than learning static functions.

In related work, Lakin *et al.* [89] designed and simulated a system based on enzymatic chemistry, capable of learning linear functions of the form $k_1 x_1 + k_2 x_2$. Compared to the AASP, the system lacks cross-weight competition, meaning the weights cannot formally represent negative numbers, and so the system could model only strictly additive functions with $k_1, k_2 \geq 0$. Besides the regular inputs x_1 and x_2, the AASP utilizes also the bias (constant shift), hence it can model linear functions of a more general form $k_1 x_1 + k_2 x_2 + k_0$ as well as nonlinear (quadratic) functions of the form $k x_1 x_2 + k_0$, where $k_1, k_2, k_0 \in \mathbb{R}$. The AASP uses 18 reactions, however, by excluding the bias (k_0) part, it would need just 13 as opposed to 27 reactions used in Lakin's system. On the other hand, Lakin's system targets a specific wet implementation based on deoxyribozyme chemistry, so the higher number of reactions is justifiable. Last but not least, we evaluated the performance more precisely over 10,000 instead of 10 trials.

6

DELAY LINE

In current chemical reaction networks, it is difficult to coherently store and retrieve values as we are used to in traditional computer architectures. To build more complicated systems in chemistries, the ability to look at data in the past would be a valuable tool to perform complex calculations. Here, we propose a specific kind of memory applied widely in computer and electrical engineering, a delay line, also called a shift register. A delay line buffers the past inputs over a sliding window and presents them for reading (consumption) both sequentially but also as a parallel output. In our implementations enzymes acting as phase signals are a means to facilitate the copy reactions and consequently reduce the error.

Once implemented in a wet chemistry, a delay line could have significant applications in the areas of smart medication and biochemical computing [9, 61, 92, 116, 156, 168]. Rather than having a fixed dosage of a specific type of medicine, a patient could be observed over a time window and then adjust the drug release (in quantity or species) to best respond to the body's needs [102, 103]. With a time delay line, the detection would not be limited to a static response based on the actual chemical state, but could be extended to measure a chemical concentration as time series as well as capture at what point the event occurred. Temporally-enabled chemical learners, such as chemical perceptrons introduced in Chapter 4 and 5, could be used for signal memorization, processing, and

prediction to act as proactive, adaptable (and programmable) agents in human bodies.

This work presents three types of chemical delay lines—manual signalling, backpropagation signalling, and parallel-accessible. The manual signalling delay line relies on a manual injection of the copy signals to indicate when it is time to shift the values. The second model features an automatic backward propagation of the copy signals, where only the bottom-most signal needs to be injected. Although the functioning of these two models is simple and minimizes the number of species and reactions, which is linear in the number of buffered values, they provide solely a sequential access to the content and their inherent latency results in poor scalability. The most advanced delay line, the parallel-accessible delay line, addresses these issues and achieves optimal performance by employing wait queues and operating on the basis of two alternating signals (catalysts).

The system's modularity allows for integration with existing chemical systems. We illustrate the delay line capabilities by connecting the parallel-accessible delay line with an analog asymmetric signal perceptron (AASP) as presented in Chapter 5. In our setup, a parallel-accessible delay line feeds an underlying AASP with past input concentrations, and therefore allows to solve temporal tasks. We evaluated the performance of our new delayed AASP on four temporal tasks: the linear weighted moving average, the moving maximum, and two variants of the *Nonlinear AutoRegressive Moving Average* (NARMA). We also scale the system's size to more than two cached inputs to study the effect of memory on learning.

This work has been published in parts in [19, 114] and has been done in collaboration with Josh Moles.

6.1 MODEL

A chemical delay line stores past input concentrations in a queue and provides them to a connected system. It is essentially a memory that allows the underlying system such as a chemical learner to look at and utilize the past inputs (Figure 6.1).

We present three variants of a chemical delay line—manual signalling, backpropagation signalling, and parallel-accessible. The core (shared) set of species is as follows. The species X represents the externally injected input value of the delay line. A delay line maintains two copies of each (past) input, X_i and X_i^C, which (ideally) hold the concentration of input X provided before $i - 1$ iterations. The terminal species X_i is available for consumption to the underlying system, and another copy X_{i+1}^C is buffered and propagated to the next stage. Besides the species X, X_i, and X_i^C each delay line employs the signal species, which act as catalysts (triggers) of the copy reactions. The signals propagate the cached values recursively deeper in the hierarchy.
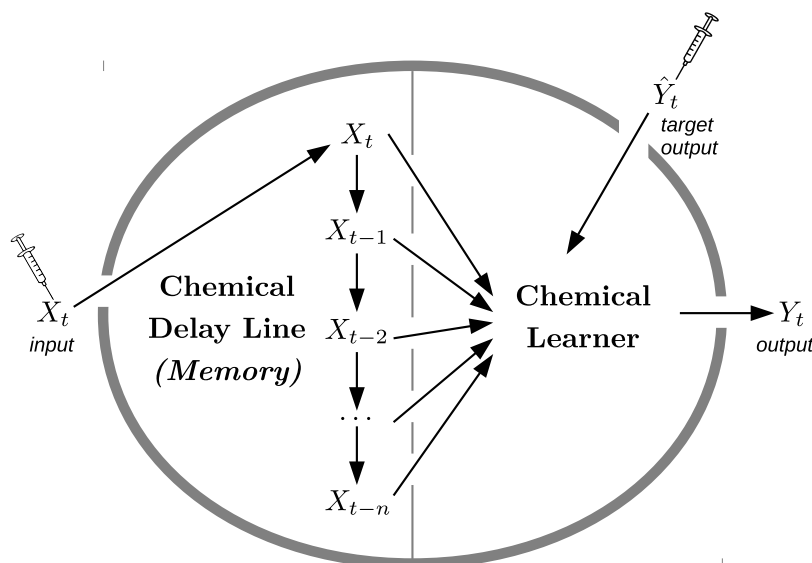
Figure 6.1: Delay line as a chemical learner's memory.

6.1.1 Manual Signalling Delay Line

The manual signalling delay line uses a specific signal X_i^S, which splits the cached input X_i^C (or injected input X) into the terminal X_{i+1} and the cached copy X_{i+1}^C propagated to the next stage. The operation of this model is fully sequential and relies on injections of signals X_i^S for each copy phase in a backward order. The reactions of the manual signalling delay line are

$$X_i^C \xrightarrow{X_i^S} X_{i+1} + X_{i+1}^C$$

$$X_i^S \rightarrow \lambda,$$

<div align="right">(6.1)</div>

where $X_0^C = X$ is the injected input. Each copy phase, triggered by an injection of the associated signal X_i^S, starts only when the previous copy phase is completed, i.e., when the shared buffered species X_{i+1}^C is fully consumed. The rate of a signal decay $X_i^S \rightarrow \lambda$ is
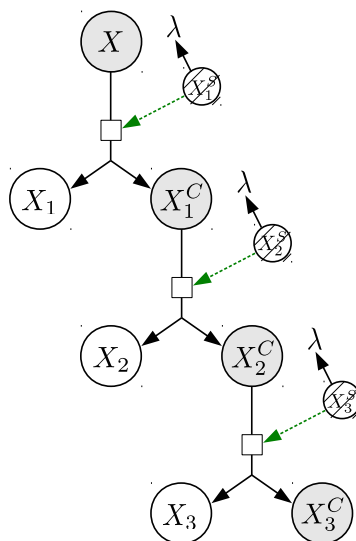
Figure 6.2: The manual signalling chemical delay line of size $n = 3$. The species X_1, X_2, and X_3 represent the input X cached at time/cycle $t, t-1$, and $t-2$ respectively. The manual signalling model relies on sequential injection of the signals X_3^S, X_2^S, and X_1^S, hence it produces the cached values one after another. Nodes represent species, solid lines are reactions, dashed lines are catalysts, and λ stands for no or inert species.

set such that a time window in which the signal X_i^S is present is sufficient to fully catalyze its copy reaction.

In the delay line with three stages (as shown in Figure 6.2) an injection of the signal X_3^S produces X_3 from cached X_2^C, then the signal X_2^S copies (splits) the cached X_1^C into X_2 and X_2^C, and finally the signal X_1^S copies the input X into X_1 and X_1^C. The terminal species X_1, X_2, and X_3 representing the actual and past inputs are consumed. Figure 6.3 shows a propagation of values for the three-staged manual signalling delay line in finer detail.

Separation and sequentiality of the copying stages results in error-free functioning of the delay line, however, this model requires a significant amount of external "help" since the number of signal injections equals the number of stages (cached values) of the system.

6.1.2 Backpropagation Signalling Delay Line

The backpropagation signalling delay line keeps the basic copy (cleave) mechanism of the manual version but treats the signal species differently. More specifically, only the bottom-most signal X_n^S, where n is the number of stages, needs to be injected. The signals react and transform in a bottom-up chain. This process is governed by the delay line's reactions, therefore no external help is needed. The reactions of the backpropagation signalling delay line are

$$X_i^C \xrightarrow{X_i^S} X_{i+1} + X_{i+1}^C$$

$$X_{i+1}^S \rightarrow X_i^S \qquad \qquad \boxed{6.2}$$

$$X_1^S \rightarrow \lambda,$$
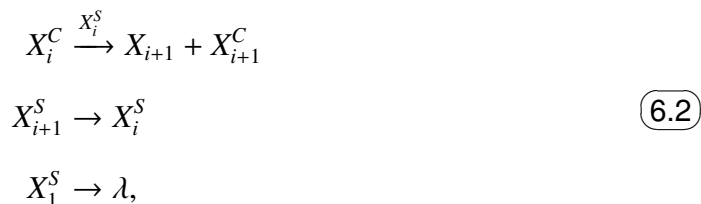
where $X_0^C = X$ is the injected input. Note that the top signal X_1^S is still removed by a decay as in the manual variant. The advantage of this model is that the user is required to perform only two injections, of an input and the bottom-most signal, at the beginning of

(a) t_0 (b) t_1 (c) t_2 (d) t_3 (e) t_4

(f) t_5 (g) t_6 (h) t_7 (i) t_8 (j) t_9

(k) t_{10} (l) t_{11} (m) t_{12} (n) t_{13} (o) t_{14}
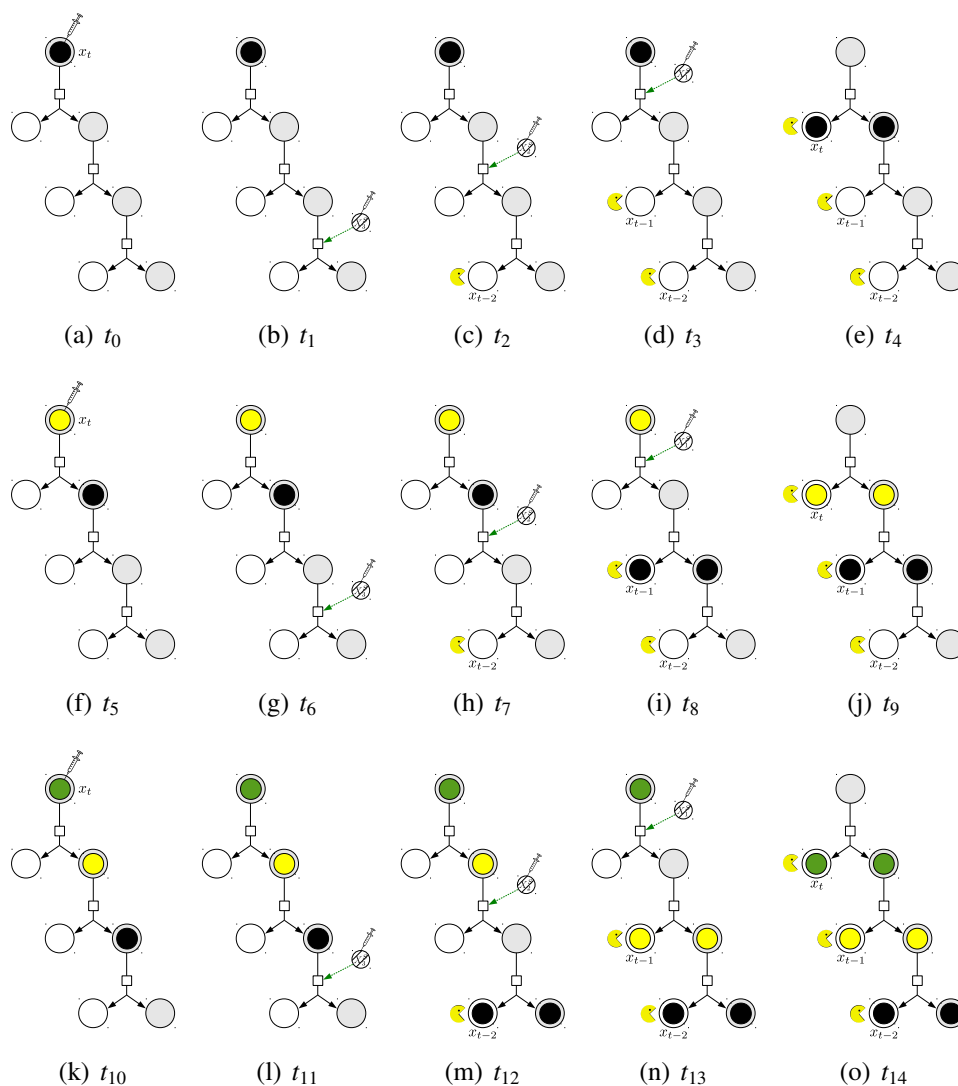
Figure 6.3: Diagrams illustrating a propagation of the past input values in the manual signalling chemical delay line of size $n = 3$. The input concentrations injected at time t_0, t_5, and t_{10} are shown as black, yellow, and green circles respectively. The signals injected at time t_1, t_6, and t_{11} (X_3^S), t_2, t_7, and t_{12} (X_2^S), and t_3, t_8, and t_{13} (X_1^S), trigger the copy reactions, which produce the terminal species consumed by the underlying system. Note that because the signals operate sequentially, so do the production of terminals. Consequently, the latency of the system grows linearly with the number of cached values.

the cycle and then the system transforms the species internally. Despite an adaptation of the signalling, the main characteristic of the manual model, i.e., a sequential access and consequently a linear latency, applies also for this variant.

Figure 6.4 shows the delay line with three stages, where an injection of the signal X_3^S produces X_3 from cached X_2^C. Simultenously the signal X_3^S is transformed to X_2^S and consequently to X_1^S splitting the cached X_1^C into X_2 and X_2^C, and the input X into X_1 and X_1^C. Figure 6.5 shows a propagation of values for the three-staged backpropagation signalling delay line in finer detail.

Since the reactions performing signal backpropagation are not instantaneous, there is always an overlap of consecutive stages, where both signals X_{i+1}^S and X_i^S are present. This overlap allows the partial parallelism of this system. In this time window the shared buffer species X_{i+1}^C is simultaneously consumed by the reaction catalyzed by X_{i+1}^S and produced by the X_i^S's reaction, i.e., there is effectively a direct path from X_i^C to cascade down to X_{i+2}^C (and X_{i+2}). As a matter of fact, a portion of the more current cached input X_i^C might "leak" to X_{i+2}^C prematurely. That produces an error that accumulates with every stage.
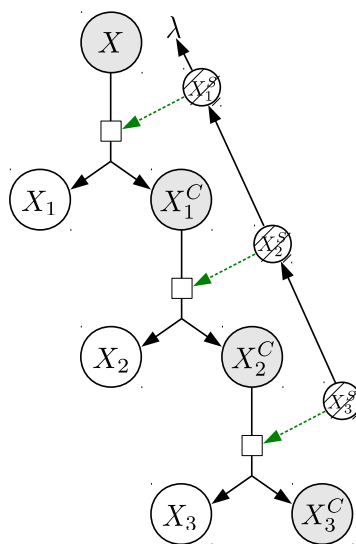
Figure 6.4: The backpropagation signalling chemical DL of size $n = 3$. The species X_1, X_2, and X_3 represent the input X cached at time/cycle $t, t-1$, and $t-2$ respectively. This model is semi-sequential (semi-parallel) since the occurrence of signals X_3^S, X_2^S, and X_1^S, which are transformed one from another backwards, partially overlap. Nodes represent species, solid lines are reactions, dashed lines are catalysts, and λ stands for no or inert species.

Table 6.1: Maximal and average copy error of the backpropagation signalling delay line calculated as SAMP for 10,000 runs with 200 iterations each. The rate constants were optimized by genetic algorithms for each size independently.

Size	Max	Average
2	2.28%	1.97%
3	5.26%	4.84%
4	11.66%	11.25%
5	14.35%	14.09%

By choosing rate constants carefully we can minimize the time window in which the signals are simultaneously present. This helps to minimize the error. To optimize the rate constants we employed standard genetic algorithms. Table 6.1 shows the overall copy error for the best solutions found. Since the error grows rapidly with the system size this model could be used only for small sizes, such as $n = 2$ or $n = 3$. Depending on the desired properties of the delay line, this is worth considering for the application. More detailed information can be found in [114].

6.1.3 Parallel-Accessible Delay Line

As opposed to blocking sequential stages employed before, the parallel-accessible delay line (PDL) executes several non-concurrent stages in parallel. It is the final optimized variant of a chemical delay line with a minimal error, minimal latency, minimal number of injections, and a constant number of signals. The reactions of the parallel-accessible delay line are

$$X_i^C \xrightarrow{S} X_{i+1}^{T_1} + X_{i+1}^C \qquad S = S_1 \text{ \textit{iff} } i \text{ is even, } S = S_2 \text{ otherwise}$$

$$X_i^{T_j} \xrightarrow{S} X_i^{T_{j+1}} (j < i) \qquad S = S_1 \text{ \textit{iff} } i + j - 1 \text{ is even, } S = S_2 \text{ otherwise}$$

$$S_1 \to \lambda$$

$$S_2 \to \lambda,$$
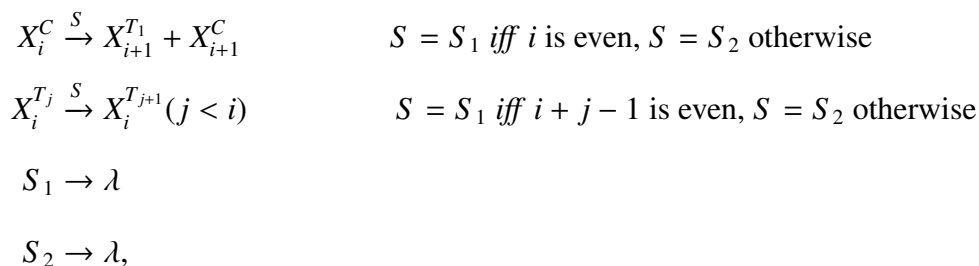
(6.3)
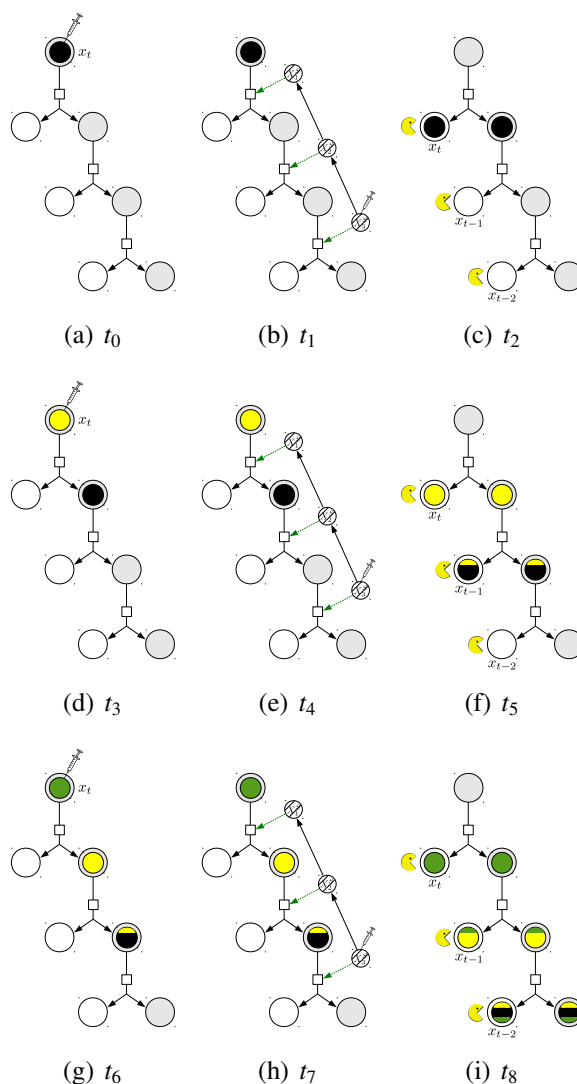
Figure 6.5: Diagrams illustrating a propagation of the past input values in the backpropagation signalling chemical delay line of size $n = 3$. The input concentrations injected at time t_0, t_3, and t_6 are shown as black, yellow, and green circles respectively. The signal X_3^S injected at time t_1, t_4, and t_7 transforms to the signals X_2^S and X_1^S, which trigger the copy reactions producing the terminal species consumed by the underlying system. Note that because of the partially overlapping stages the neighboring copy reactions leak a portion of the cached values prematurely. That produces an error that accumulates with every stage (illustrated with color stripes).

where $X_0^C = X$ is the injected input. The parallel-accessible delay line combines the standard copy reactions $X_i^C \xrightarrow{S_1/S_2} X_{i+1} + X_{i+1}^C$ with wait queues $X_i^{T_j} \xrightarrow{S_1/S_2} X_i^{T_{j+1}}$, where the past concentrations are buffered for the required number of cycles. Only two alternating

signals S_1 (red) and S_2 (blue) are needed to drive the delay line's execution, regardless of its size. The key idea here is that the signals S_1 and S_2 alternate in catalyzing the reactions, therefore, all evenly indexed (and then all oddly indexed) reactions could be executed at the same time without a conflict. Out of each pair of two neighboring species, there is always at least one with zero concentration since no S_1 (or S_2) signals drive the adjacent reactions.

As shown in Figure 6.6, an injection of S_1 fires a simultaneous production of all the values X_1, X_2, and X_3, and other copy or move-in-wait-queue reactions. The signal S_2 injected some time after S_1 triggers the remaining reactions that move values further to terminals X_i, consumed by an underlying system. Note that the concentration pathway from X to X_i contains $2i - 1$ reactions with i red and $i - 1$ blue signals. Figure 6.7 presents a propagation of values for the three-staged parallel-accessible delay line in finer detail.
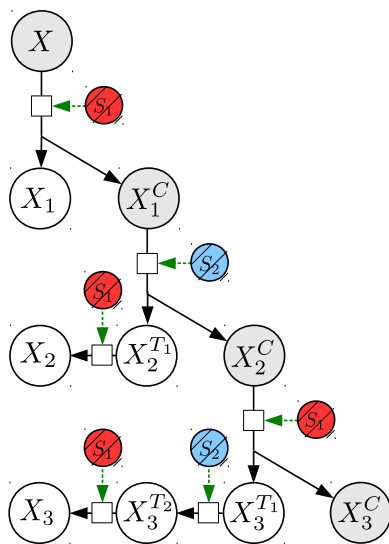
Figure 6.6: The parallel-accessible delay line of size $n = 3$. The species X_1, X_2, and X_3 represent the input X cached at time/cycle $t, t - 1$, and $t - 2$ respectively. The parallel model utilizes a wait queue X_i^T for each stage and two alternating signals S_1 and S_2, which produces all the values simultaneously. Nodes represent species, solid lines are reactions, dashed lines are catalysts, and λ stands for no or inert species.
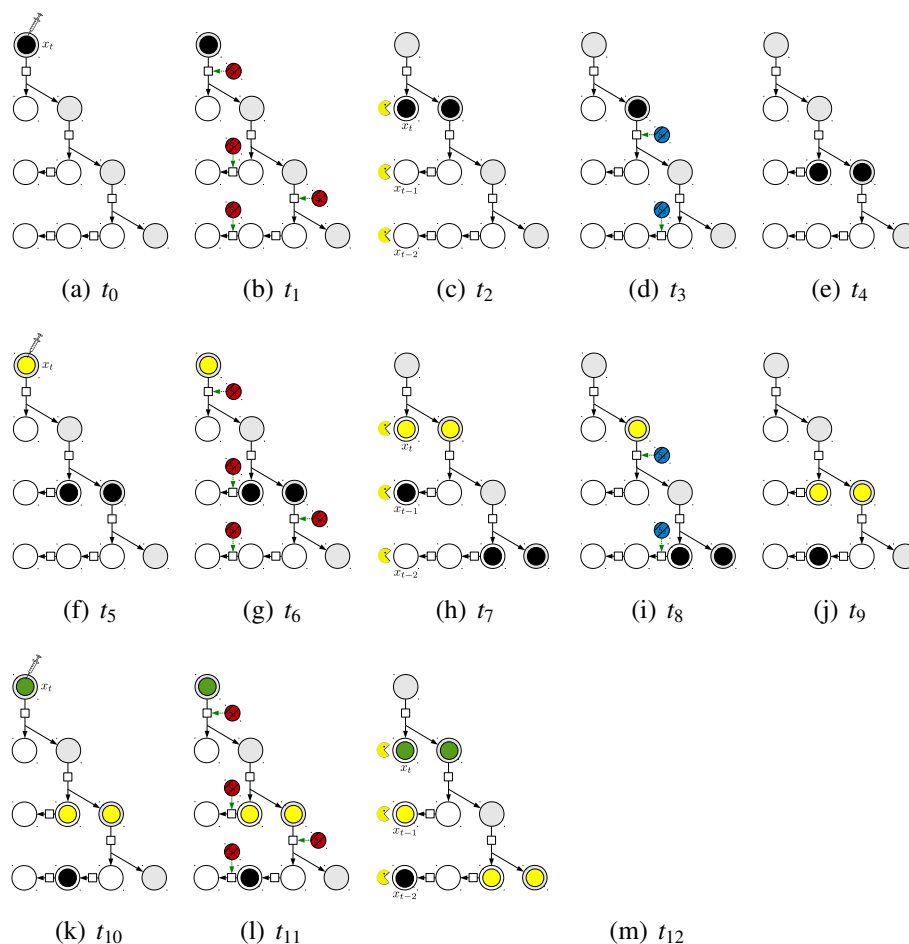
Figure 6.7: Diagrams illustrating a propagation of the past input values in the parallel-accessible chemical delay line of size $n = 3$. The input concentrations injected at time t_0, t_5, and t_{10} are shown as black, yellow, and green circles respectively. The red signal S_1 injected at time t_1, t_6, and t_{11}, and the blue signal S_2 injected at time t_3 and t_8 trigger the copy and move-in-wait-queue reactions, which produce all the terminal species consumed by the underlying system at the same time.

Due to a negligible error and a constant latency, i.e., all past values are available immediately, the PDL could easily integrate with other chemical systems and provide them with a reliable and fast-access memory. Compared to our previous models, the new model, however, requires more species and reactions. Because of the wait queues, the number grows quadratically for both the species $\frac{(n+2)(n+1)}{2}$ and the reactions $\frac{(n+1)n}{2}$. Table 6.2 summarizes the attributes of all our chemical delay line models.

Table 6.2: Comparison of three different types of a chemical delay line.

Attribute	Manual DL	Backprop. DL	Parallel DL
Error	0	exp	0
Injections	$O(n)$	$O(1)$	$O(1)$
Latency	$O(n)$	$O(n)$	$O(1)$
Signals	$O(n)$	$O(n)$	$O(1)$
Species	$O(n)$	$O(n)$	$O(n^2)$
Reactions	$O(n)$	$O(n)$	$O(n^2)$

6.2   PERCEPTRON INTEGRATION

In this section we will integrate an AASP (Chapter 5) with a PDL to create a new memory-enabled AASP (AASP-DL) by feeding the species X_i produced by the PDL directly into an AASP as shown in Figure 6.8. This setup will allow us to perform temporal signal processing effectively. We set the rates of all PDL reactions to $k_{cat} = 2$ and $K_m = 0.075$. In order to fully proceed the S_1 phase in 25 steps, but at the same time prevent a signal overlap that could produce a transfer error, we set the decay rate of S_1 and S_2 to 0.6.

In order to operate with arbitrary memory we extend the AASP to more than two inputs. For each new cached input X_i we add five reactions: $X_i \xrightarrow{W_i} X_i^L + Y$ and $X_i + Y \to \lambda$ for the input-weight integration, and $W^\oplus \xrightarrow{X_i^L} W_i$, $W^\ominus \xrightarrow{X_i^L} W_i^\ominus$, and $W_i + W_i^\ominus \to \lambda$ for the positive and negative weight adaptation. The AASP of size n has therefore $4n + 9$ species and $5n + 8$ reactions. All new reactions use the same rate constants as the original X_1/X_2 reactions. Since the values X_i are produced rapidly, the AASP's timing, such as the injection of the input signal S_{in}, are unchanged. Also, a single DL operational cycle of 50 time steps is compatible with 500 time steps required for the AASP's training cycle. To demonstrate the scalability, we model AASP-DLs of size 2 to 5.
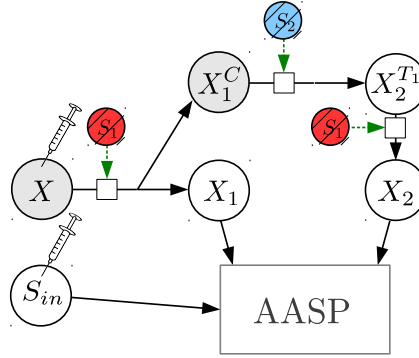
Figure 6.8: An AASP-DL schematic of size $n = 2$.

6.3 EXPERIMENTS

In this section we describe the tasks, the performance metrics, the training setup, and the learning results of the AASP-DL.

6.3.1 Tasks

The selection and setting of temporal tasks reflect the fact that the expected AASP's output concentration must be between the minimal (zero) and the maximal output concentration, which is equal to the sum of all inputs provided $[S_{in}] + [X_1] + \ldots + [X_n]$. The tasks used to evaluate the performance of our new system follow.

1) LWMA2: The Linear Weighted Moving Average (LWMA) is a time series of a lagged averages, where each past element is weighted by an arbitrary value. The LWMA of order 2 is defined as

$$y_t = k_1 u_{t-1} + k_2 u_{t-2} + k_0, \qquad (6.4)$$

where $k_1, k_2 \in (0.2, 0.8)$ are randomly drawn constants, $k_0 \in (0.1, 0.4)$ is a constant bias, and u_t is an *i.i.d* input stream generated uniformly from $(0.2, 1)$. The task is to output y_t based on the past inputs u_{t-i}.

2) WMM2: The Weighted Moving Maximum (WMM) is a time series of maximum lagged inputs. The WMM of order two is defined as

$$y_t = k \max(u_{t-1}, u_{t-2}) + k_0, \qquad (6.5)$$

where similarly to LWMA2 the constants k and k_0 are randomly drawn from the interval $(0.2, 0.8)$ and $(0.1, 0.4)$ respectively, and u_t is an i.i.d input stream generated uniformly from $(0.2, 1)$. The task is to output y_t based on the past inputs u_{t-i}.

3,4) NARMA: The Nonlinear AutoRegressive Moving Average (NARMA) [15] is a discrete time series, where the current output y_t depends on both the previous inputs and outputs up to a given depth (order). The NARMA task of order n is defined as

$$y_t = \alpha y_{t-1} + \beta y_{t-1} \sum_{i=1}^{n} y_{t-i} + \gamma u_{t-n} u_{t-1} + \delta, \qquad (6.6)$$

where $\alpha = 0.3, \beta = 0.05, \gamma = 1.5, \delta = 0.1$, and u_t is an i.i.d input stream generated uniformly from an interval $(0, 0.5)$. The task is to produce the output y_t based on the previous inputs u_{t-i}. NARMA is widely used as a benchmark task in the neural and reservoir computing literature [78, 99] due to its nonlinearity and dependence on long-term memory.

We tackle two variants, NARMA2 and NARMA10, i.e., the second and tenth order of the problem. Since NARMA10 can be unstable, we bound the series by a non-linear tanh saturation function. Also, we scale the target stream y_t for NARMA2 by 2 to better fit the AASP's output range.

6.3.2 Performance Measures

We define performance by two standard error measures: the *symmetric absolute mean percentage error* (SAMP) with values ranging from 0% to 100%

$$\text{SAMP} = 100\langle\frac{|y - \hat{y}|}{y + \hat{y}}\rangle, \tag{6.7}$$

and the *root normalized mean square error* (RNMSE)

$$\text{RNMSE} = \sqrt{\frac{\langle(y - \hat{y})^2\rangle}{\sigma_{\hat{y}}^2}}, \tag{6.8}$$

where the square error is normalized by $\sigma_{\hat{y}}^2$, a variance of the target output $\hat{y}$. A RNMSE of 1 therefore corresponds to chance level.

6.3.3 Training Setup

The training of all AASP-DLs starts with a random setting of weight concentrations from $(0.5, 1.5)$. During each training iteration we first inject the input X with a concentration corresponding to u_t for all tasks. Then we set the bias input S_{in} concentration to 0.5 for LWMA2 and WMM2, and 0.1 for NARMA2 and NARMA10. To trigger the DL operation we also provide the signal S_1, which immediately produces both the current and the cached values X_i. To finish a PDL buffering procedure, we inject another DL signal S_2 25 time steps later. Then again after 25 time steps, we finally provide both the learning signal S_L and the target output $\hat{Y}$ representing y_t to initiate the weight adaptation.

We run the AASP-DL against 10,000 training time series of length 800 for each task. We then evaluate the RNMSE and SAMP performance over 10,000 runs for each training iteration, however, we report only the final training error.

6.3.4 Results

The AASP-DL reaches a relatively small error for all temporal tasks, which demonstrates that even a single chemical perceptron posses sufficient learning and computing capabilities. The error of the AASP-DL with optimal size settles to the range of $(0.10, 0.77)$ for RNMSE and $(1.20\%, 7.37\%)$ for SAMP as shown in Figure 6.9. Figures 6.10 and 6.11 show RNMSE and SAMP for all tasks over time.

The easiest function is a linear LWMA2 (RNMSE of 0.10), with performance decreasing as memory of the past inputs grows. That is to be expected because LWMA2 depends only on the last two inputs u_{t-1} and u_{t-2}, so any additional information is essentially superfluous. Note that the AASP cannot fully eliminate the contribution or consumption of an extra input-weight branch, hence the input here basically acts as a noise. Figure 6.12 shows an example of the weight concentration traces and the filtered output and target output for LWMA2.

The WMM2 task's output is also fully prescribed by the last two inputs, however as opposed to LWMA2, the AASP-DL of size 5 performs best (RNMSE of 0.32) with
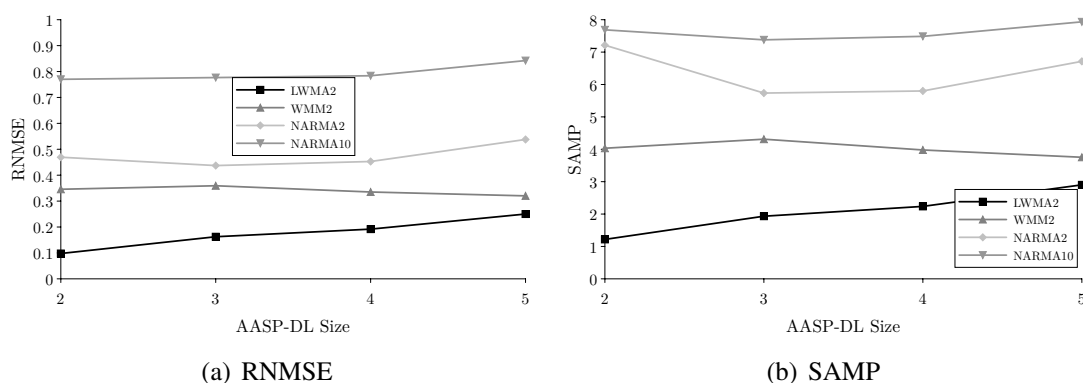
Figure 6.9: The relation between the final RNMSE and SAMP error and the AASP-DL size after 800 learning iterations. For each task 10,000 runs were performed.

a marginal difference to the $n = 2$ instance (RNMSE of 0.35). Even though the extra past inputs do not affect the target output y_t, the AASP might utilize them as a statistical variance source.

Because of its recurrent definition, the NARMA2 performance improves significantly for a longer DL, reaching an optimum for $n = 3$ (RNMSE of 0.44) and slightly worse for $n = 4$ (RNMSE of 0.45). Since the NARMA2 depends on the last two inputs u_{t-1} and u_{t-2} and recurrently on the last two outputs y_{t-1} and y_{t-2}, its output is fully determined by the last four inputs, which is inline with our results. NARMA10 is the most difficult task (RNMSE of 0.77) due to the function dependence on long lags. The performance is fairly constant for $n \leq 4$.

(a) LWMA2

(b) WMM2

(c) NARMA2

(d) NARMA10
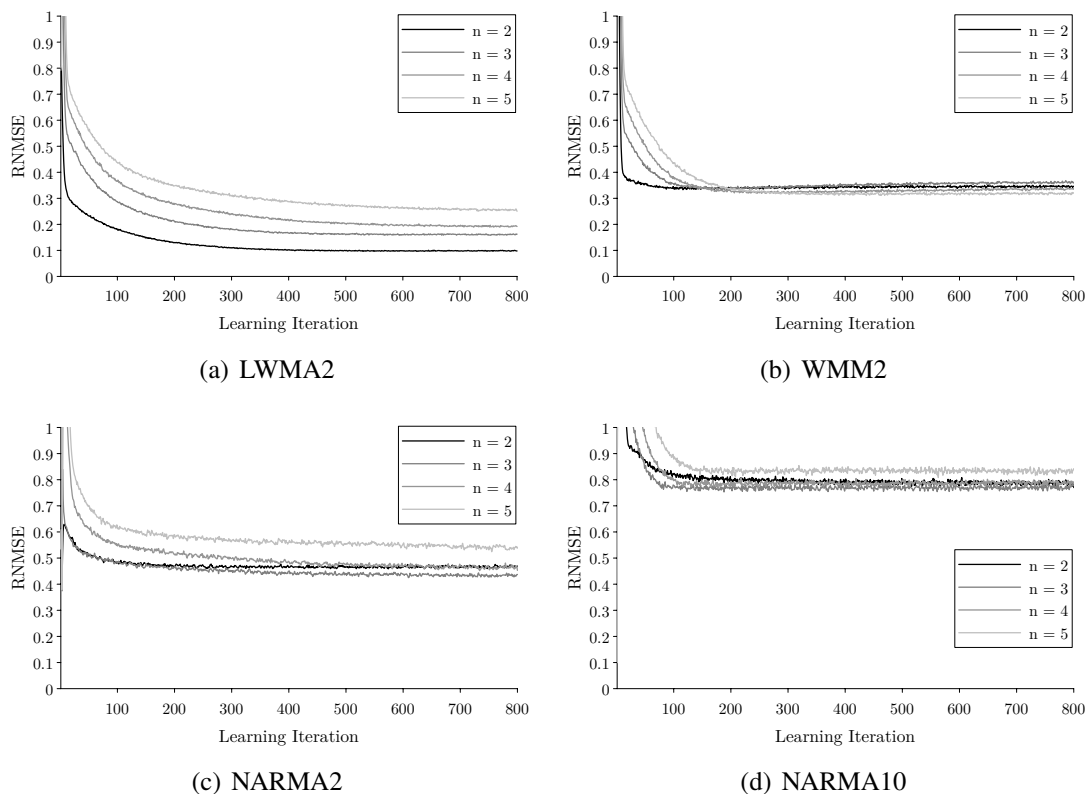
Figure 6.10: The RNMSE error over time for all tasks. Average values over 10,000 runs.

(a) LWMA2

(b) WMM2

(c) NARMA2

(d) NARMA10
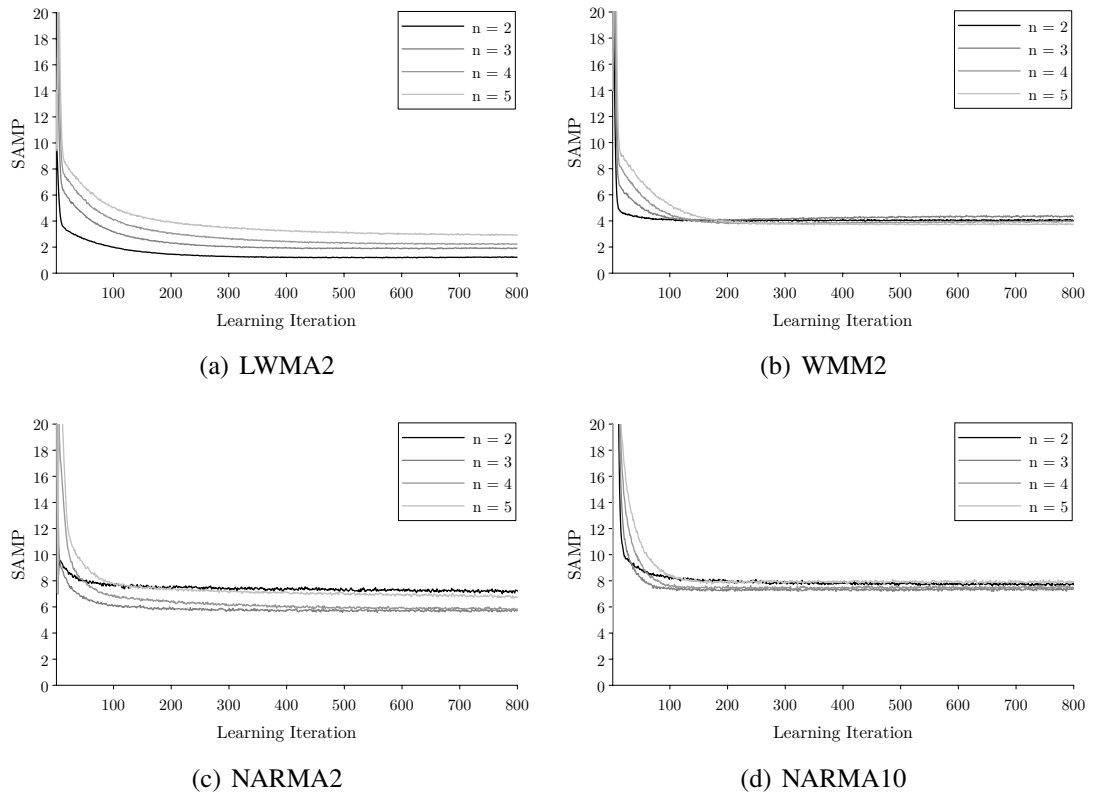
Figure 6.11: The SAMP error over time for all tasks. Average values over 10,000 runs.
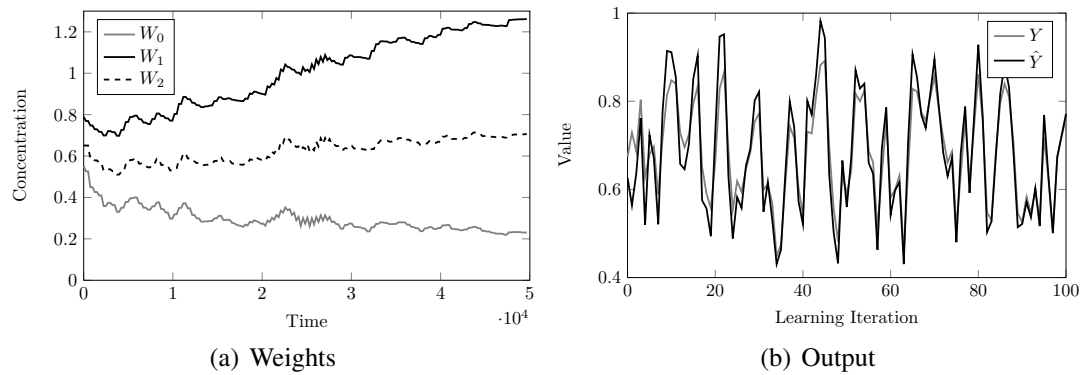
(a) Weights

(b) Output

Figure 6.12: An example of the AASP-DL $n = 2$ learning LWMA2, showing (a) the concentration traces of the weights and (b) the filtered output and the target output.

6.4 DISCUSSION

In this chapter we demonstrated that the idea of chemical learning can be extended to temporal tasks by utilizing a memory of past concentrations provided through a chemical delay line. The AASP-DL successfully learns to produce a weighted moving average as well as a moving maximum. These operations could be applied to monitoring certain substances in a patient's blood, such as the insulin level, and perhaps also be used to control it by a conditional release of a specific amount of a required substance (cure). To demonstrate more complex nonlinear time dependencies, we also trained the AASP-DL for both the NARMA2 and NARMA10 benchmark tasks.

Our results show that memory improves learning for the recursive NARMA2 and nonlinear WMM2 tasks, but leads to lower performance for the simple LWMA2 task. Because the weights compete with each other and the AASP cannot fully eliminate the contribution or consumption of an extra input-weight branch, for a memory larger than the task inherently requires, performance decreases. Note that for the initial weights drawn uniformly from the interval $(0.5 - 1.5)$, the AASP's ideal output region optimized by genetic algorithms (Chapter 5) is around half of the maximal output concentration, which equals the total amount of input injected $[S_{in}] + [X_1] + [X_2]$. To move the input-output relation closer to that region, we scaled the NARMA2 task by two.

Our work reported in [114] presents an integration of the backpropagation signalling delay line with a thresholded asymmetric signal perceptron (Chapter 4). The temporally-equipped binary chemical perceptron successfully learns 14 linearly separable binary functions over a sliding window of size two.

In related work Jiang *et al.* [81] introduced the concept of a delay element. The delay element is primarily used as a storage area for holding data in between each computation

cycle. The data then returns and is examined in computing during the next iteration of the calculation. Jiang's buffer is primarily a signal processing application looking only at the previous value. Our delay line has the ability to delay not only multiple steps in time, but also allows access to any of the past values besides the most recent. Note that we could create a FIFO [83] out of the delay line by removing the intermediate stages and providing only the final output.

Other areas, such as networking, use chemical reaction networks as a mechanism to control scheduling and queuing of packets [107]. The work discusses a methodology to use the law of mass action as a means to schedule packets. With our buffer Meyer's systems could also be extended to actually implement a means to queue packets in a chemical system. This would reduce cost and complexity by having a single implementation medium.

7

MULTICOMPARTMENT FEEDFORWARD CHEMICAL NEURAL NETWORK

Having developed a family of individual chemical perceptrons, we wish to design a method for connecting these in a more computationally-powerful network and investigate their composability. The network should be modular, such that networks with different topologies are constructed from different combinations of the same parts. As neural networks have been shown to be powerful machine learners, we hope that a network of single chemical perceptrons could be a step towards the first general-use reprogrammable (retrainable) chemical computer.

We achieve this goal with the *Feedforward Chemical Neural Network* (FCNN), a network of cellular compartments, each containing a chemical neuron as a module. Communication between nodes in the network is achieved by permeation through the compartment walls, facilitating the network's feedforward and backpropagation mechanisms.

Like standard single-layer perceptrons, each of our individual chemical perceptrons can learn the linearly separable binary two-input logic functions, such as AND and OR, but they are incapable of learning the linearly inseparable functions XOR and XNOR [110]. Here, we demonstrate that the FCNN learns each of these functions. To our knowledge, it is the first chemical system able to learn a linearly inseparable function.

The FCNN presented here has the simplest feedforward neural network topology: one hidden layer with two neurons, the same as the first classical neural net to learn

XOR via backpropagation [133]. For more complex learning problems, we show how the FCNN's modular design can be applied to topologies with more, or larger hidden layers. In any case, using an FCNN is as simple as injecting a few species into its outermost chemical container, and measuring the concentration of an output species. By modulating the concentrations of injected species, different inputs can be provided, the network can be trained to new tasks, and its learning rate can be annealed.

Built on a realistic model of chemical dynamics, the FCNN is a step towards reusable, reprogrammable chemical computers. We hypothesize that the FCNN, or something very similar, will likely form the basis of the first full-featured neural networks to be built in a purely chemical medium. Bringing computation into the chemical domain will not only change medicine, but computer science and biology as well. By implementing human-programmed computation in biochemical settings, we will also be a step closer to understanding the information-processing that has always occurred at the cellular level of life. The FCNN shows that a relatively simple chemical reaction network can learn and perform complex computation.

This work has been published in parts in [27] and has been done in collaboration with Drew Blount.

7.1 CELLULAR COMPARTMENTS

Since Aristotle, Pasteur, and more recently Varela and Maturana [104], cellular compartmentalization has been considered a central characteristic of living systems. The analogy of the cell as a self-contained, self-sustaining regulatory machine is a familiar one, and the most basic requirement for such a machine is compartmentalization—separation from the outside world. Several important philosophical explanations of biological organiza-

tion consider this concept of closure to be essential [39]. Hence, much effort has gone into determining the salient characteristics of a cell, and simulating cells in the field of artificial life [51, 75, 139, 148]. For these reasons, cells have been a common component of several significant accomplishments in chemical computing [24, 43, 150].

In our chemical system, we use cellular walls as containers for the individual chemical perceptrons, which compose the FCNN. Interaction between neurons is facilitated by rules of permeation across the membranes of these cells. We refer to these permeation rules as *channels*.

Channels can be either reactive or inert. In the first case, a species is allowed to enter the cellular wall, but a different species emerges on the other side. Chemically, we imagine a molecule which reacts with the cell wall as it passes through it. A species passing through an inert channel is not changed; it simply travels from one side of the membrane to the other. A given cell wall can have any number of channels of either type.

Unlike the reactions in our CRN, whose kinetics are driven by simple but chemically accurate equations of mass-action and Michaelis-Menten kinetics, there is no obvious and simple choice for a model of permeation kinetics. In nature, cell walls, membranes, biofilms, and similar structures exist in a number of different forms. There are numerous models describing the permeability of these structures in a variety of different contexts and levels of detail [84, 87, 117, 139].

In modelling membrane permeation, it is common to consider the pressure inside the cell, or, in a similar vein, the numbers of particular molecules within it [84, 139]. In the latter case, a chemical species S permeates into a cell more slowly if there is already a high concentration of S in the cell. Furthermore, if a cell's total volume is constrained, species can permeate into it only up to a certain point, when the cell becomes 'full.' More detailed models also consider solute permeability, viscosity, hydraulic conductivity [87],

pH, and temperature.

We make several simplifying assumptions about our cells. First, the cell's internal pressure is constant, meaning either that the total volume of permeation is much smaller than that of the cell, or that the cell's volume can dynamically expand and contract to maintain pressure. Second, permeation rules are inherently one-way; if a species passes from side *A* of a cell wall to side *B*, it does so aware only of the state on side *A*. This means that permeation is not osmotic, or equilibrium-seeking. Third, since we consider chemical species only in terms of their concentrations, other physical parameters such as temperature and viscosity are beyond the scope of our model. Having thus simplified the chemical picture, channel permeation rates follow mass-action kinetics: the rate of permeation is exactly proportional to the concentration of the source species—in the source container—and a permeability constant *k*.

We introduce a notation for channels. Consider two example channels through the wall **C**,

$$\mathbf{C} : (S_2 \leftarrow S_1), \text{ and}$$
$$\mathbf{C} : (S_1' \rightarrow S_2').$$

$\boxed{7.1}$

The first and second species within the parenthesis are always inside and outside of **C**, respectively, and the arrow denotes the direction of permeation. Here $\mathbf{C} : (S_2 \leftarrow S_1)$ is a reactive channel in which S_1 passes into **C**, turning into S_2 in the process.

Cells and their membranes are the central object of study in the field of P Systems (also called membrane systems), which considers heavily abstracted chemical systems where different reactions occur in different cells, with communication between cells via membrane permeation [124, 125]. The work presented here is superficially similar to P Systems, so it is important to understand key differences between the two models. Our

approach to chemistry is less abstracted than that taken in P Systems: the chemical objects of P Systems are strings that 'react' through rewriting rules akin to Chomsky's context-free grammars [124]. P Systems successfully demonstrated the computational power of formal grammars of a specific type, but significant omissions, such as any kinetic model, separate them from chemical computing in practice.

Our CRN [48, 71] constitutes a logic built upon the ordinary differential equations. Our task is to construct autonomously learning neural networks from systems of these ODEs, so our chemical perceptrons are mathematically distinct from perceptrons as they are commonly defined formally [62].

Each phase of the operation of a multilayered perceptron, such as 'calculate linear sum,' and 'update weights,' is only qualitatively emulated by our ODEs—we do not aim to reproduce the mathematics of neural networks in a one-to-one fashion in chemistry. Moreover, chemical reactions representing each operation run continuously and in parallel, rather than discretely and in sequence. Because our network is a large, nonlinear system of ODEs, there are generally no analytic solutions for, say, what concentration of the output Y will be present in our system a set time after it is started with given initial conditions. We therefore use numerical ODE solvers as the backbone of our simulations, as discussed in Section 7.5.

Previous results on the theoretical power of perceptrons and neural networks (e.g., Hornik's proof that feedforward perceptrons are universal approximators [73]) start with the mathematical definitions of the models, and thus rely upon assumptions (as basic as $y = \sum w_i x_i$) that are not valid in our chemical perceptrons. Furthermore, as discussed in Section 7.1, our networks are restricted to tree-like topologies, a restriction which is generally not made in the study of classical perceptrons.

Nonetheless, Rumelhart's classic exposition of backpropagation [133] treats just our

special case: solving XOR with a one-hidden-layer, two-hidden-unit feedforward percep-
tron. In Section 7.5 we will use this early result from classical multilayer perceptrons as
a benchmark against our FCNN.

7.2 CHEMICAL NEURONS

The chemical neuron, which forms the basis of the FCNN is the Analog Asymmetric Sig-
nal Perceptron (AASP). Two new variants of the AASP are used, one for hidden neurons
and one for the output neuron, discussed in Section 7.2.2. Having discussed the neu-
rons, we move on to the network: we specify the compartments containing each single
perceptron and the channels between them in Section 7.3. It is through these permeation
channels that the signal feeds forward through the network, and the error propagates back-
ward. As a reference throughout this section and the rest of the chapter, see Table 7.1,
which lists all chemical species in the FCNN. Tables listing all reactions, reaction rate
constants, and permeation channels are provided in the Appendix.

7.2.1 Our Chemical Neuron: The AASP

The AASP forms the basis of our network. Chapter 5 introduced the AASP and described
its mechanism and motivation in depth. Because both our hidden and output neurons
(described in Sections 7.2.2 and 7.2.2) are modified versions of the AASP, we describe
the relevant features of the AASP to the FCNN in this section.

Input

Each component of the input vector is represented by a species X_i. Though the AASP
accepts continuous input values, it also operates very well as a binary-input perceptron.

Table 7.1: Chemical Species in the FCNN
*: OCN only; #: HCN only

Function	Species
Inputs	X_1, X_2
Output	Y
Weights	W_0, W_1, W_2
Input (clock) signal	S_{in}
Learning signal	S_L
Production records	X_1^L, X_2^L, S_{in}^L
Weight adjusters	$W^\oplus, W^\ominus$
Indiv. weight decreasers	$W_0^\ominus, W_1^\ominus, W_2^\ominus$
Inert input transmit *	X_1', X_2', S_{in}'
Binary threshold *	T
Penalty *	P
Error signal *	$E^\oplus, E^\ominus$
Backprop signals *	$P_1^\oplus, P_1^\ominus, P_2^\oplus, P_2^\ominus$
Feedforward signal #	S_F
Feedforward output #	F

In this case, we inject only those X_i whose value is 1, at a preset input concentration, and do not inject the X_i whose value is 0.

Under this input encoding, the zero input ($X_i = 0$ for all i) corresponds to no chemical injection, which poses problems for a system that should react to such an input—the zero input is indistinguishable from no input at all. We therefore include a special clock signal S_{in} with every input, alerting the perceptron to the input event. Though S_{in} is necessary only to recognize the zero input, it is included with all inputs for the sake of consistency. We will see later that S_{in} is also useful in the weight integration.

Input-Weight Integration

Like a formal perceptron's weights, our chemical perceptrons' weights determine the persistent state of the system, and when adjusted, modulate the mapping from input to output. With reactions between the weight species W_i, the input species X_i, and the output

species Y, we wish to qualitatively reproduce the simple linear sum,

$$y = w_0 + w_1 x_1 + w_2 x_2, \qquad \boxed{7.2}$$

in such a way that it reproduces the functionality of negative weights.

What we require of each W_i is simply that, in the presence of X_i, the output Y is either increased or decreased depending on the concentration of W_i. This is achieved by a race between two reactions that consume X_i. In the first, W_i catalyzes X_i's transformation into Y. In the second, X_i and Y annihilate.

$$X_i \xrightarrow{W_i} Y + X_i^L$$

$$X_i + Y \rightarrow \lambda \qquad \boxed{7.3}$$

Note that the first reaction, which produces Y as a function of X_i and W_i, simultaneously produces a record-keeping species X_i^L. This species is later used in the weight-update stage of learning, as will be described in Section 7.2.1. Since the clock species S_{in} is already present in every injection, it acts as the constant-one coefficient of the bias W_0. In terms of equation 7.3, $S_{in} = X_0$.

Recalling the reaction rate laws, we see that $[Y]$'s rate of change $d[Y]/dt$ during the input-weight integration is the sum of two terms for each input:

$$\frac{d[Y]}{dt} = \sum_i \left(\frac{k_{c,i}[X_i][W_i]}{K_{m,i} + [X_i]} - k_{a,i}[X_i][Y] \right), \qquad \boxed{7.4}$$

where each k is a reaction rate constant. The terms inside the sum are the rates of the reactions in Equation 7.3; their signs are opposite because one reaction produces Y and the other consumes it. Because the first reaction's rate is proportional to $[W_i]$, a large $[W_i]$ will result in the first reaction producing Y faster than the second reaction consumes it.

In this case, the net effect of the two reactions is to increase $[Y]$. When $[W_i]$ is small, the second reaction dominates and $[Y]$ decreases. Thus, the concentration of the weight species W_i determines if the presence of X_i serves to increase or decrease the final output.

Note that upon input injection, each W_i, X_i pair is simultaneously producing and annihilating the same Y. A consequence of this is another disanalogy between chemical and formal perceptrons: weight strengths are relative, as each copy of the second reaction above will proceed at a rate proportional to $[Y]$, which is constantly changing in a manner dependent on every input weight.

Note that there is no analytic solution to $[Y]$ in the above equation, hence our use of numeric ODE solvers, discussed in Section 7.5. Though we cannot determine the nonlinear dynamics of $[Y]$ without running an ODE solver, the nature of the chemical reaction network enforces a lower bound of zero and an upper bound equal to the sum of the input concentrations, $\sum_i [X_i]$. The upper bound is asymptotically reached as weights go to infinity.

Learning

Chemical implementations of the input-weight integration are not new [66, 85], but our previous work was the first to implement autonomous learning [21]. Though each of our neurons has a slightly modified learning process from the AASP, explaining the AASP's will lay the groundwork for learning in chemical neurons. In an AASP, learning starts with the injection of a desired output species $\hat{Y}$, and ultimately updates each weight in the appropriate direction by a magnitude relative to that input dimension's impact on the most recent output Y and an analog of error.

The first step of the learning process is reading the output of the AASP. After input injection, the integration reactions are allowed to run for a preset period of time, chosen

so that the injected input species are fully consumed. In our simulations, this lasted 50 time steps. At the end of this period, the concentration of Y in the AASP is read as its output.

The next step in the AASP's learning process is determining the correctness of this output. This is accomplished by injecting the desired output species $\hat{Y}$ at the concentration we desire of Y. Upon this injection, Y and $\hat{Y}$ quickly annihilate each other in equal proportions via the reaction,

$$Y + \hat{Y} \to \lambda. \tag{7.5}$$

This reaction consumes all of whichever species has the lower initial concentration, and the remaining species' concentration will be equal to the difference in initial concentrations, i.e., it will be an analog of the error. We then use whichever species remains to create weight-changing species with the appropriate sign, in a slower reaction than the above (to ensure something akin to 'execution order'):

$$\hat{Y} \to W^{\oplus}, \text{ and}$$
$$Y \xrightarrow{S_L} W^{\oplus}, \tag{7.6}$$

where the learning-signal species S_L, also injected with $\hat{Y}$, ensures that Y is transformed into $W^{\oplus}$ only after $\hat{Y}$'s injection. This process, by which the output is compared with the desired output to produce weight-adjustment species. Similar processes are used to calculate error and weight-adjustment signals in both the hidden and output neurons in the FCNN.

Once weight-adjustment signals are produced, the AASP, and the two modified version introduced in Section 7.2.2 all behave identically, adjusting their i^{th} weight in proportion to both the adjustment signal and the ith input dimension's influence on the most

recent production of Y. This qualitatively reproduces the so-called delta rule of classic perceptron learning [62].

In Section 7.2.1, we introduced the species X_i^L, which is produced as a record of the impact of X_i and W_i on the production of Y. Via catalysis, we use this species to emulate the delta rule: the weight-adjustment species $W^\oplus$ and $W^\ominus$ adjust the concentration of weight W_i through productive and annihilatory reactions, respectively, each catalyzed by X_i^L. Thus, weight-adjustment is achieved by the reactions

$$W^\oplus \xrightarrow{X_i^L} W_i,$$

$$W^\ominus \xrightarrow{X_i^L} W_i^\ominus, \text{ and} \qquad \boxed{7.7}$$

$$W_i^\ominus + W_i \to \lambda.$$

The main difference between this method and the classical delta rule is that, because X_i^L's production is influenced positively by W_i, and X_i^L also catalyzes W_i's adjustment, larger W_i will be adjusted relatively more than smaller ones. Thus, larger weights are adjusted more than smaller ones. We reproduce the effect that the ith weight is adjusted proportionally to both the difference between desired and actual output, and the most recent i^{th} input signal.

Note that it takes an intermediate species and two reactions for $W^\ominus$ to annihilate W_i. This is simply because the hypothetical reaction $W^\ominus + W_i \xrightarrow{X_i^L} \lambda$, with two reactants and a catalyst, would require the collision of three molecules in the same instant. As such three-molecule interactions are unlikely, we do not use them in our designs.

7.2.2 Two Breeds of AASP

To accommodate communication between neurons in FCNNs, we had to modify the original AASP design. This resulted in two related breeds: one for hidden neurons, which

has a modulated output schedule and accepts backpropagated error signals; and another for the output neuron, modified to initialize the cascading backpropagation reactions. We discuss the means by which these neurons are connected to each other to achieve feeding-forward and backpropagation in Sections 7.3.2 and 7.3.3, but here we first discuss the details that distinguish our output and hidden neurons from each other and the AASP.

The Hidden Chemical Neuron

The *Hidden Chemical Neuron* (HCN) is the modular component from which FCNNs of various topologies are constructed. It has two differences from the AASP as presented in Section 7.2.1, one each to facilitate feeding forward and backpropagation.

The AASP produces output Y constantly, as the input dimensions are gradually integrated through the series of reactions in Section 7.2.1. It is designed to receive input signals instantaneously, however, so in a network context, we cannot simply feed forward a hidden neuron's output as it is produced. We have in practice found that AASPs perform poorly with their input injected gradually over time.

Thus, we introduce a *feedforward species S_F*, which arrives in an HCN's chemical system when its output Y is meant to feed forward. The details of this will be discussed along with the rest of the network in Section 7.3.2, but for now it is enough to know that the following reaction occurs in each HCN:

$$Y \xrightarrow{S_F} F, \qquad (7.8)$$

where F is the species that is later fed forward. Thus the next neuron receives an input signal that is more sudden than the relatively gradual output of the HCN, and this action is induced by S_F.

In terms of learning, an HCN has less work to do than an AASP. An AASP reasons about its output and a desired output to produce the weight-update species $W^{\oplus}$ and $W^{\ominus}$, but hidden neurons receive error signals through backpropagation, not through a direct comparison of the outputs. Thus, the $W^{\oplus}$ and $W^{\ominus}$-producing reactions of an AASP are omitted from the HCN.

The Output Chemical Neuron

The *Output Chemical Neuron* (OCN) has a different learning mechanism than the AASP. As the current FCNN is designed to learn binary functions, such as XOR, inserting a desired output species to instigate learning is somewhat ineffective. For example, if the binary threshold is 0.5, the error will be minuscule if the actual output is, say 0.499. This was not a serious problem in the single AASP, but the OCN's error signal must not only update its own weights, but propagate backwards as well. The reactions involving backpropagation will be discussed in Section 7.3.3; here, we explain the method by which weight-changing signals are produced within the OCN.

The OCN's learning phase begins with the external evaluation of its output. A penalty species P is injected only if the OCN's output is incorrect, i.e., on the wrong side of the binary threshold that is used to interpret chemical concentrations as binary values. Along with P, the AASP's learning signal S_L is injected. Further, a threshold species T is injected in concentration equivalent to the binary threshold. T 'communicates' the binary threshold to the OCN, and behaves somewhat analogously to the 'desired output' species $\hat{Y}$ in the AASP (Section 7.2.1).

The goal of the OCN's error-evaluating reactions, diagrammed in Figure 7.1, is to amplify small continuous-valued errors to emulate distinct binary-valued errors. This is done in two stages. First, Y and T annihilate in a fast reaction. Whichever is left
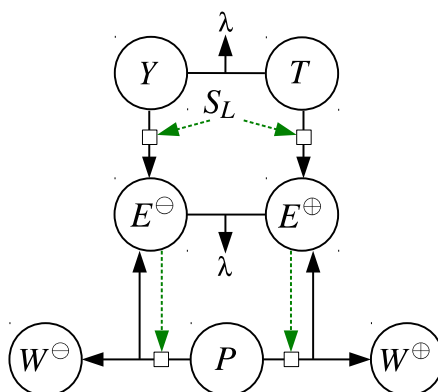
Figure 7.1: The weight-update mechanism for a two-input AASP. The process is started by the injection of penalty signal P. The anihilatory comparison of the output Y and the threshold T determines whether the weights will be increased or decreased. Nodes represent species, solid lines are reactions, dashed lines are catalysts, and λ stands for no or inert species.

over encodes whether Y was above or below the threshold, and so if weights should be increased or decreased. So S_L catalyzes relatively slow reactions from Y and T to signed error species $E^\ominus$ and $E^\oplus$, respectively.

Whichever E species is more prevalent tells whether the weights should be increased or decreased, but their absolute magnitudes might be very small. We then amplify these signals auto-catalytically while consuming the penalty species P:

$$
\begin{aligned}
P &\xrightarrow{E^\oplus} E^\oplus + W^\oplus, \\
P &\xrightarrow{E^\ominus} E^\ominus + W^\ominus.
\end{aligned}
$$

(7.9)

Note that $E^\oplus$ ($E^\ominus$) is both catalyst and product, and $W^\oplus$ ($W^\ominus$) is also produced. The above equations are illustrated in the bottom half of Figure 7.1.

The autocatalytic reactions ensure that the total amount of weight-change is not limited by the initial difference between $[Y]$ and $[T]$, which is encoded in the $[E]$s. The total amount of weight-change is bounded, however, by the injected concentration of P, as it is the only reactant creating $W^\oplus$ or $W^\ominus$. Thus, we can achieve annealing by decreasing the

concentration of successive injections of P. To summarize the error-evaluating reactions: the initial difference between $[Y]$ and $[T]$ determines the sign of the weight adjustment and P determines the magnitude.

7.3 NETWORKING

This section discusses the methods by which AASPs are connected to make a FCNN. We first describe the network's topology, both as a neural network and as a series of chemical containers, then its mechanisms of feeding forward and error backpropagation.
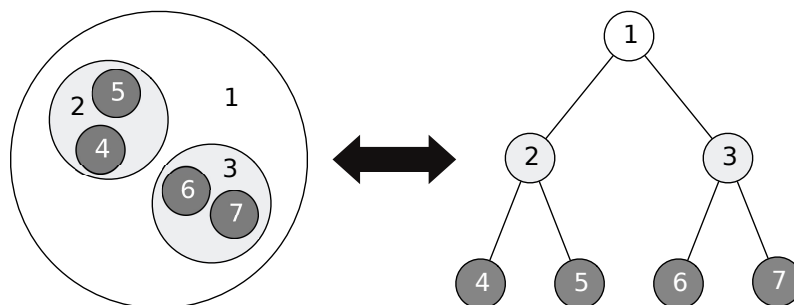
Figure 7.2: FCNNs have tree-like topologies.

7.3.1 Constructing the Network Topology

With neurons in nested cells and links across cell walls, our networks are topologically trees, with each wall a branch and the outermost cell the root (Figure 7.2). The outermost cell (1 in the figure) corresponds to the output layer in a feedfoward multilayer perceptron, and for a given set of nested cells, the deepest cells (4-7 in the figure) correspond to the input layer. We cannot construct arbitrary feedforward networks, as in our tree-like networks each node can feed forward to only one node in the next layer.

This nested architecture crucially enables the FCNN's modularity. Each neuron is guaranteed to share a permeable wall with any neuron to which it is connected in the

network, so messages which must pass from one neuron to another do not need to be 'addressed'—they passively permeate from parent to child or vice-versa. This allows for scalability. The signal species between different pairs of neurons can be chemically identical because the signalling occurs in distinct compartments. For this reason, the number of species in the FCNN is sub-linear in the size of the network.

If we wished to make arbitrary feed-forward topologies, it would be possible only if we included distinct signal species for every linkage in the neural network. This could be achieved by placing all hidden neurons, regardless of their level in the network, as sibling subcompartments within the larger output neuron. Feedforward and backpropagation signals would travel between hidden neurons via this shared compartment. As long as there are unique signal species for each link in the network, this design allows for arbitrary feedforward network topologies.

Here we utilized a simple two-hidden-neuron, one-hidden-layer topology to solve XOR. This topology is consistent with either of the above paradigms. As we are more interested in the modularity afforded by a tree-like topology, we implement backpropagation and feeding forward in ways generalizable to tree-like designs.
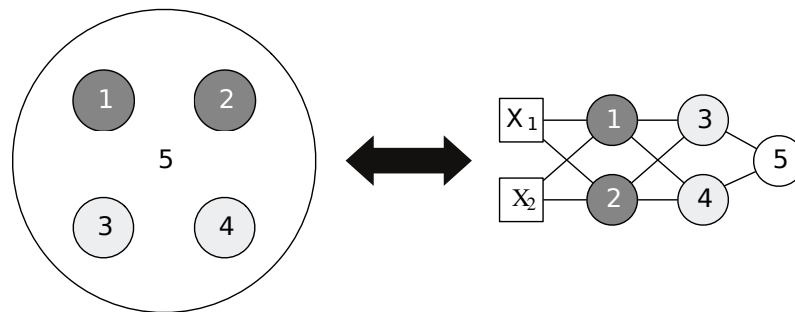
Figure 7.3: An alternative scheme places all hidden neurons as sibling subcompartments of the output neuron, illustrated for an all-to-all network topology.
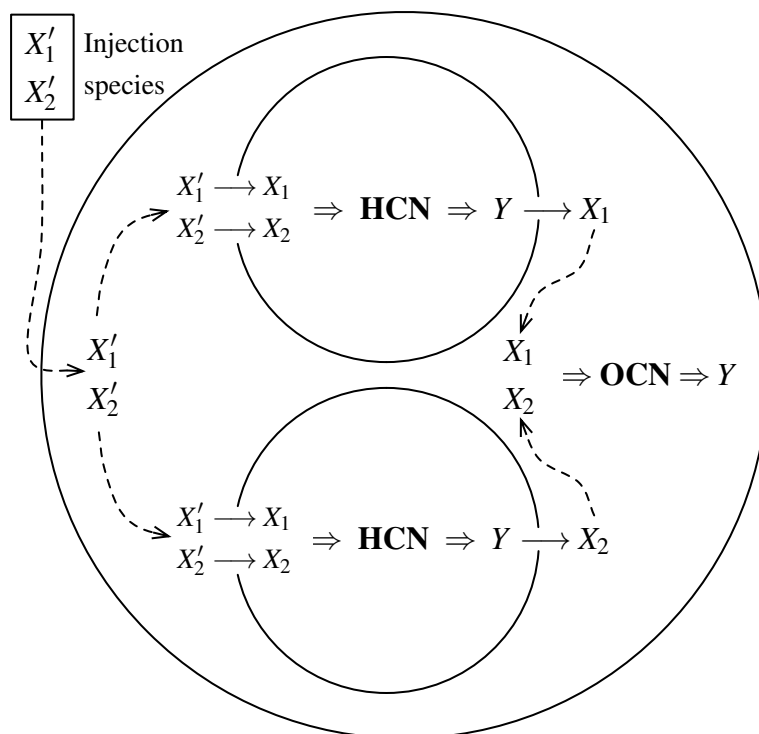
Figure 7.4: A simplified diagram of the feedforward action of a two-input, one-hidden-layer, two-hidden-neuron FCNN. The inert X_i' species are injected into the outer layer and permeate into the input layer, turning into the reactive X_i input species in the process. Each inner compartment produces Y, which then permeates into the outer compartment as it is transformed into the appropriate X_i. This feedforward process is modulated by unshown species S_F and F, see Section 7.2.2.

7.3.2 The Forward Pass

Injection

Each of an FCNN's input neurons is contained in a subcompartment of the output neuron. To correspond easily with wet chemistries, all injections into the system are made directly into the outermost compartment, rather than precisely into any of its subcompartments. Yet the input species must be consumed only by reactions in the input-layer nodes, and because the output compartment contains an AASP, any input species X_i are automatically consumed whenever they are present within it. Therefore, an inert species X_i' is injected

instead, that permeates into the input compartments. These cells have channels which convert each inert X_i' to the reactive input species X_i. These species are immediately treated by the hidden neurons as input.

$$\mathbf{C} : (X_i \leftarrow X_i') \quad \text{for } \mathbf{C} \text{ an input cell} \tag{7.10}$$
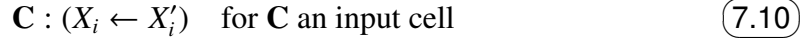
Similarly, each hidden neuron needs to receive a feedforward species S_F (described in Section 7.2.2), which must be ultimately injected from outside the system, and every neuron in a given layer should receive this signal simultaneously. In an FCNN with n hidden layers, we require a set of feedforward species $S_F^1, S_F^2, ..., S_F^n$, which transform into the basic signal species S_F when permeating into cells of the appropriate layer:

$$\mathbf{C} : S_F \leftarrow S_F^i, \quad \mathbf{\textit{C} in the ith hidden layer.} \tag{7.11}$$
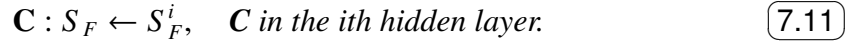
Once S_F permeates into a given hidden neuron, that neuron's output Y is transformed into the feedforward species F. If this HCN is the ith neuron, in the jth hidden layer, say it is in container ${}_j^i\mathbf{C}$. Then F permeates outward into $\mathbf{C}$, turning into the corresponding input species X_i in the process:

$$Y \xrightarrow{S_F} F$$
$$\tag{7.12}$$
$${}^i\mathbf{C} : (F \rightarrow X_i) \quad {}^i\mathbf{\textit{C} is the i^{th} subcell of \textbf{C}.}$$
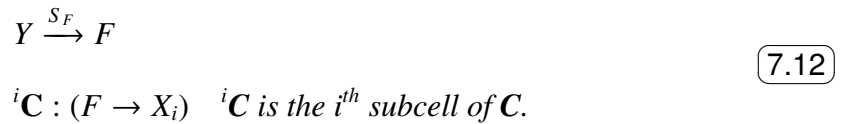
Simultaneously, the OCN receives its signal species S_{in} by recycling the HCN's S_F:

$$
{}^i\mathbf{C} : (S_F \rightarrow S_{in}) \quad {}^i\mathbf{\textit{C} is the i^{th} subcell of \textbf{C}.} \tag{7.13}
$$
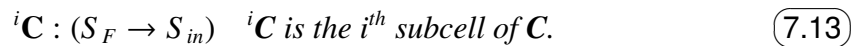
Thus, S_F plays two roles: it alerts the hidden neurons to feed forward their output,

and then by the above permeation alerts the neurons in the next layer to begin processing input. With these reactions, the output Y of each hidden chemical neuron feeds forward. This process is illustrated in Figure 7.5, which shows experimental concentration/time data for the species in the feedforward process, in each neuron in a simple one-hidden-layer FCNN.

7.3.3 Backpropagation

Backpropagation is the algorithm that first learned XOR and popularized the modern feed-forward neural network [133]. Again, our chemical system does not exactly reproduce the classic formulae defining this process, and we focus instead on chemically reproducing the important conceptual relationships between parts.

To review, classical backpropagation proceeds in three steps:

1. The output perceptron's error $e = \hat{y} - y$ is calculated, where $\hat{y}$ and y are the desired and actual outputs.

2. Moving backwards, each perceptron's error is calculated as the sum of errors which it has affected, weighted by the connecting weights. In our topologically-restricted multilayer perceptrons, each node only feeds forward to one other, so this reduces to $e_i = e_j w_{ji}$, where w_{ji} is the weight from perceptron i to j and e_j is already known.

3. Each weight is updated proportionally to this error, the weight's own magnitude, and the most recent output of the source perceptron: $\Delta w_{ji} \propto e_i w_{ji} y_i$, where y_i is the last output of perceptron i.

The first step above is emulated by the OCN's internal reactions, described in Section 7.2.2. These reactions produce weight-update species that encode the sign and magnitude of the network's overall error.
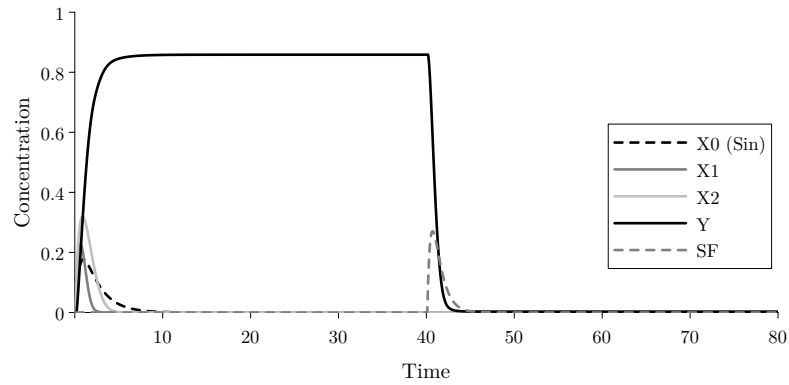
(a)
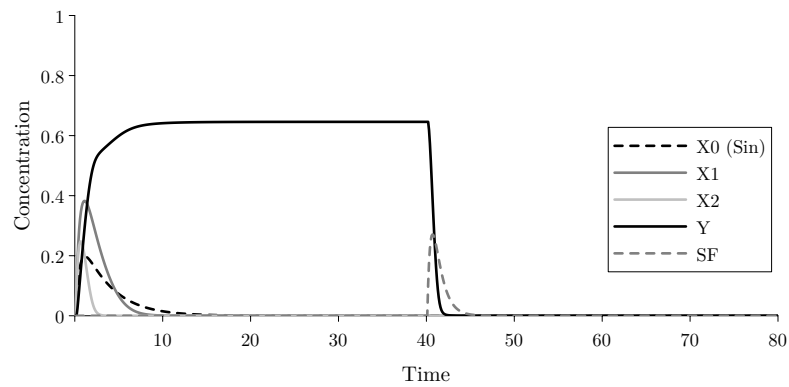
(b)
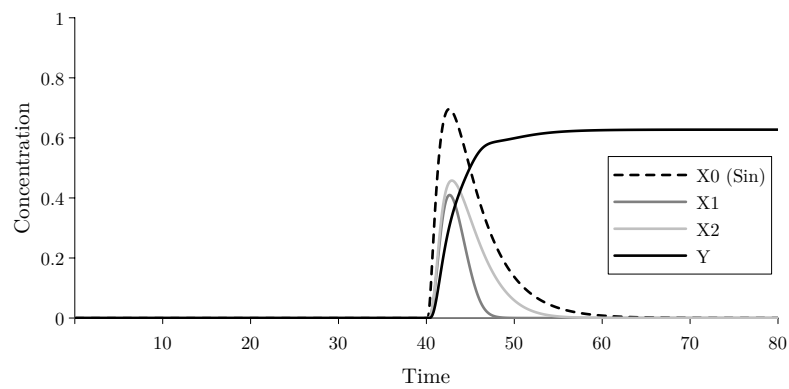
(c)

Figure 7.5: Concentration/time plot in the example FCNN with two HCNs (a, b) and one OCN (c), illustrating the HCNs' outputs feeding forward and becoming the OCN's input. At time zero the inputs X'_1, X'_2, and S'_{in} are injected to the OCN (not shown in (c)). They then permeate into each HCN, transforming into the input species. Note the initial spikes in concentrations of X_1, X_2, and S_{in} in (a) and (b). After the injection of an S_F signal at time 40, each HCN's output permeates out to the OCN, transforming into the appropriate input species and S_{in}.

125

The OCN generates input-specific error signals to be backpropagated to the previous layer in the network. These reactions are also implemented in any HCN with deeper layers of HCNs inside of it, i.e., all neurons where a signal should be backpropagated. In such neurons, the weight-adjustment species $W^\oplus$ and $W^\ominus$, in addition to changing weights, produce input-specific backpropagation signals. This production is catalyzed by the weight species W_i,

$$
\begin{aligned}
W^\oplus &\xrightarrow{W_i} P_i^\oplus, \\
W^\ominus &\xrightarrow{W_i} P_i^\ominus,
\end{aligned}
$$

$$\boxed{7.14}$$
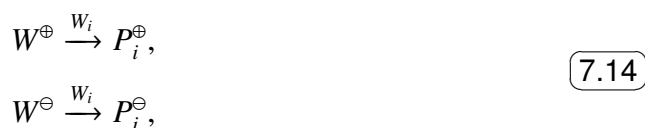
so the ith backpropagation signal $P_i^\oplus$ or $P_i^\ominus$ is produced in an amount positively correlated with the ith weight and the overall adjustment species $W^\oplus$ and $W^\ominus$.
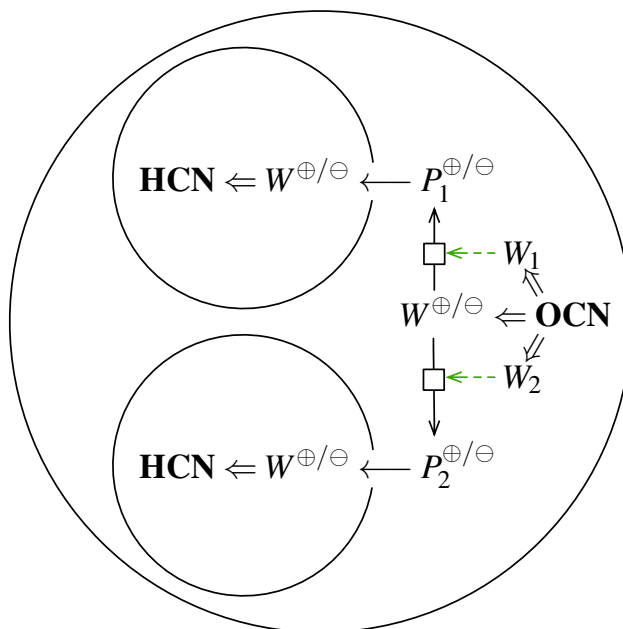
Figure 7.6: A diagram of the backpropagation action of a one-hidden-layer, two-hidden-neuron FCNN. The weight-adjusting species in the outer OCN (right of figure) produce signed, input-specific penalty species P_i. The penalty species then permeate into the hidden neurons' compartments, becoming those neurons' weight-changing species in the process.

The signed, dimension-specific penalty species $P_i^\oplus$ or $P_i^\ominus$, then propagate backwards through the network via permeation channels. Since the purpose of these signals is to adjust weights, they are simply transformed to the species $W^\oplus$ and $W^\ominus$ as they permeate into the appropriate subcontainers:

$$^i\mathbf{C} : (W^\oplus \leftarrow P_i^\oplus)$$
$$^i\mathbf{C} : (W^\ominus \leftarrow P_i^\ominus) \quad ^i\mathbf{C} \text{ is the } i^{th} \text{ subcell of } \mathbf{C}.$$

(7.15)
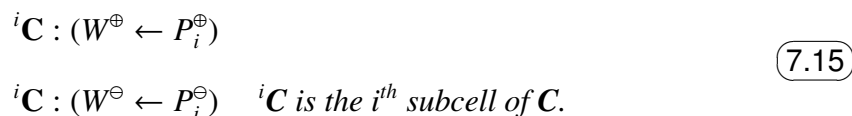
Thus, in the one-hidden-layer case, the concentration of the weight-changing species $W^\oplus$ and $W^\ominus$ in the ith inner compartment is related to a) $W^\oplus$ and $W^\ominus$ in the outer compartment, and b) the weight W_i connecting the two compartments. This relationship is shown in Figure 7.6.

7.4 METHODOLOGY

7.4.1 Rate and Permeation Constants

Though we have discussed all of the reactions and permeation channels in the FCNN, we have so far only specified the reactants, products, and catalysts in each reaction, but every reaction has at least one dimensionless rate coefficient, which describes its speed. Here we discuss how those rate coefficients have been set in our experiments. Tables listing all reaction rates can be found in Appendix B.

As each AASP (Section 7.2.1) contains around twenty distinct rate constants, the rate parameter space is prohibitively large and complex to explore via exhaustive search. In Chapter 5 we searched for these constants with a standard genetic algorithm. As both the Hidden and Output Chemical Neurons share most of their reactions with the AASP, the rates of these reactions are unmodified.

Unlike with the AASP, we found success in setting the reactions introduced in this paper by hand. Our intuition in doing so was based on the intended function of each reaction, and which reactions should occur relatively faster or slower than others. To illustrate the intuitive setting of rate constants, consider the case when two species annihilate in order to determine which had the larger initial concentration, and then the remaining species is transformed into another to perform a task dependent on the initial comparison. This is the case when Y and T are compared in the OCN's error-calculating mechanism (Figure 7.1): whichever species remains after annihilation is meant to turn into an appropriately-signed error species. In cases such as these, the comparison reaction should be faster than the follow-up reactions, dependent on that comparison. Otherwise, the second reaction would execute before the comparison had been meaningfully made. These manually set rate constants and those set by the genetic algorithm are listed in Appendix B.

7.4.2 Simulation Details

The FCNN is a large system of ODEs. As such systems are generally unsolvable analytically, we make use of numeric Runge-Kutta-type ODE solvers to observe the FCNN's behavior. We used a fixed-time-step solver with a step size of 0.1, chosen for speed as well as stability. All simulations were run on the COEL web-based chemical modelling and simulation framework, built throughout this and related projects (Chapter 10).

In the interest of modularity and flexibility of simulations, COEL's ODE solvers do not consider the FCNN as a whole. Rather, the contents of each cellular compartment are simulated semi-independently, with a separate ODE solver in each compartment and the various solvers communicating with each other as needed. This allows us to use different solvers in each compartment, or even to have compartments containing entirely different types of simulations (though this option is unused in our current applications).

When using adaptive time step-size ODE solvers, the compartments are synchronized by imposing the 'slowest' compartment's time step-size on the other compartments, and so all of the chemical concentrations in the FCNN are updated in unison. Still, this design lends itself naturally to fixed time-step solvers.

7.5 RESULTS

As our goal is to learn linearly inseparable functions in a chemical neural network, we built the first FCNN in the classic one-hidden-layer, two-hidden-neuron network topology first shown to learn XOR [133]. We will refer to this topology, with rate constants set as described in Section 7.4.1 simply as the FCNN in this section.

The FCNN's accuracy at each of the 2,000 consecutive iterations was averaged over
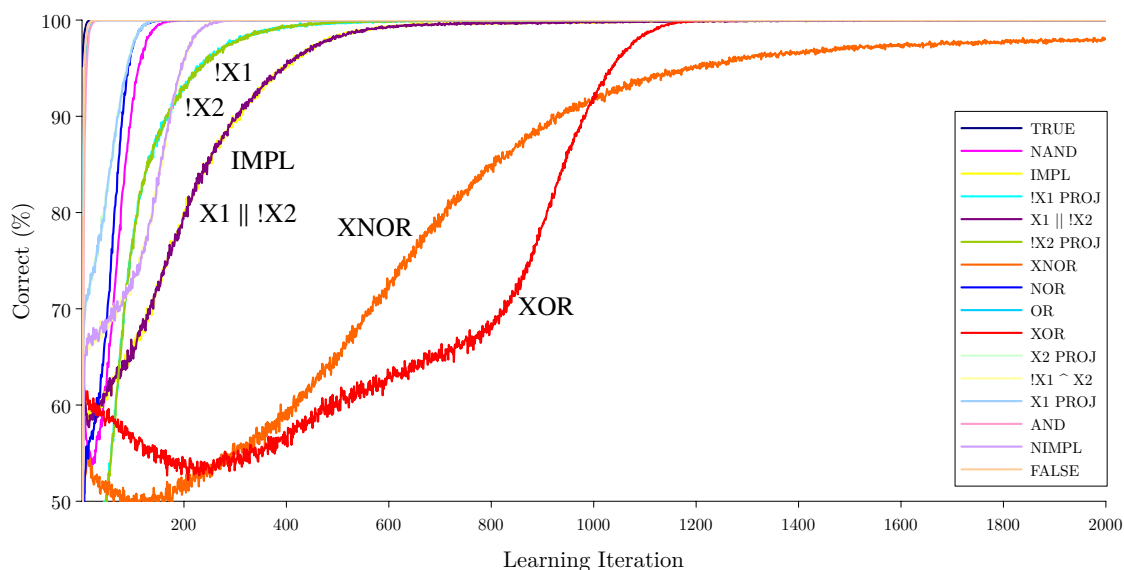
Figure 7.7: Average accuracy of the FCNN on each of the 16 binary two-input logic functions. An FCNN with one hidden layer and two hidden neurons layers was run 10,000 times on each of the 16 functions. Each run started with random initial weights and was trained for 2,000 learning iterations. The data points represent the proportion of correct answers the system produced on a given learning iteration. Six of the functions are labelled; the remaining ten overlap in the top-left of the graph. Note that the FCNN learns equally well any two functions that are equivalent after switching the names of X_1 and X_2.

10,000 training runs, to produce accuracy/time data for each logic function shown in Figure 7.7. Figure 7.8 shows an example of the FCNN converging to a solution of XOR.

The FCNN successfully learns each of the sixteen binary two-input logic functions. In a single learning run, it was trained to a given logic function with 2,000 random inputs and corresponding penalty injections (described in Section 7.3). We generated inputs randomly from $\{0, 1\}^2$ and later injected a threshold species T with concentration 0.6 and a penalty P as appropriate, 2,000 times consecutively. The concentration of the penalty species, which is analogous to a learning rate, was annealed by a factor of 0.0008 at each learning iteration. We experimentally found the best-suited annealing factor out of 10 values in the 0.0005–0.0015 range.

The most important results are the FCNN's performance on XOR and XNOR, which, because of their linear inseparability, cannot be learned by a single perceptron [110]. On the $2,000^{th}$ iteration, the FCNN's accuracy on XOR and XNOR is 100.00% and 98.05%, respectively. Averaged over all 16 functions, the FCNN is 99.88% accurate by the $2,000^{th}$ learning iteration.

As can be seen in Figure 7.7, the FCNN converges to learn inseparable functions in a relatively short period of time. The 14 separable functions are almost perfectly learned by the 600th learning iteration, but it takes until around iteration 1,200 to learn XOR. XNOR is not perfectly learned even by iteration 2,000. We see this as confirmation that linear inseparability is a challenging feature to learn. Nonetheless, the FCNN learns about as quickly as the original multilayer perceptrons that solved XOR: Rumelhart's classic multilayer perceptron took roughly 1,000 learning iterations [133].

The difference in performance between XOR and XNOR can be explained by the FCNN's asymmetric treatment of the input values 0 and 1. The functions $!X_1$ and $!X_2$ are learned almost identically well by the FCNN. This is because the FCNN architecture

is symmetric on input; each input dimension behaves according to the same logic. Its architecture is not symmetric on negation, however, as 1 and 0 values are treated fundamentally differently. Consider the fact that the output species Y is ultimately produced by the two input species X_1 and X_2, as well as the signal species S_{in} (Sections 7.2.1 and 7.3.2). For this reason, it is easier for the FCNN to learn to output 0 when it is given the
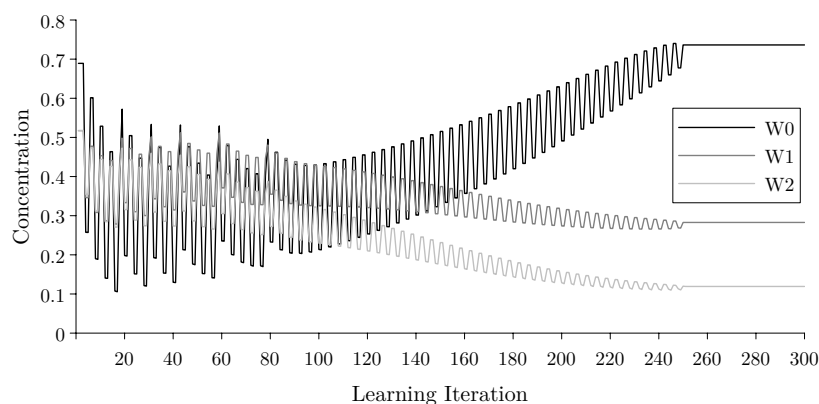
(a) Weights
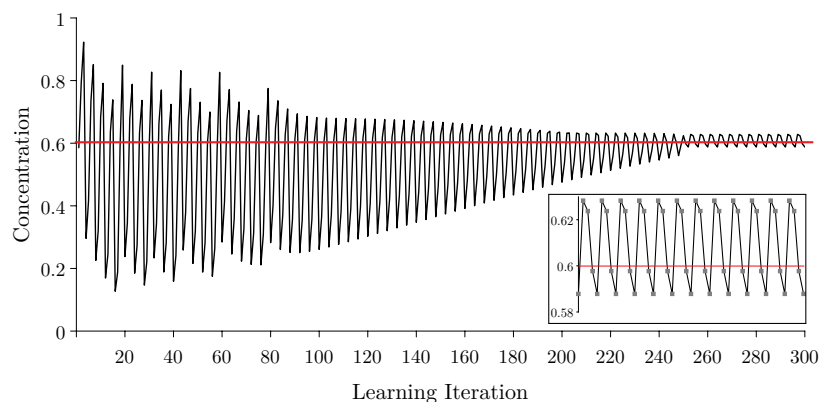
(b) Output

Figure 7.8: An example of weights and output concentration converging in the OCN as an FCNN learns XOR over 300 learning iterations. Note that when the weights reach a fixed point around the 250th iteration, the output $[Y]$ oscillates around 0.6, which in this case is the binary threshold. In this experiment, inputs were cycled in the fixed order $((0,0),(1,0),(0,1),(1,1))$ for the purpose of illustration—once the function is learned, $[Y]$ oscillates as the system produces the correct (thresholded) output stream $0,1,1,0$ (zoomed in the smaller plot).

Table 7.2: Accuracy of FCNN vs. single binary chemical perceptrons.

Accuracy	FCNN	WLP	WRP	SASP	TASP
XOR	100.00	57.61	61.88	50.53	59.00
XNOR	98.05	57.79	61.12	51.02	57.86
16-function average	99.88	94.71	95.18	93.40	94.80
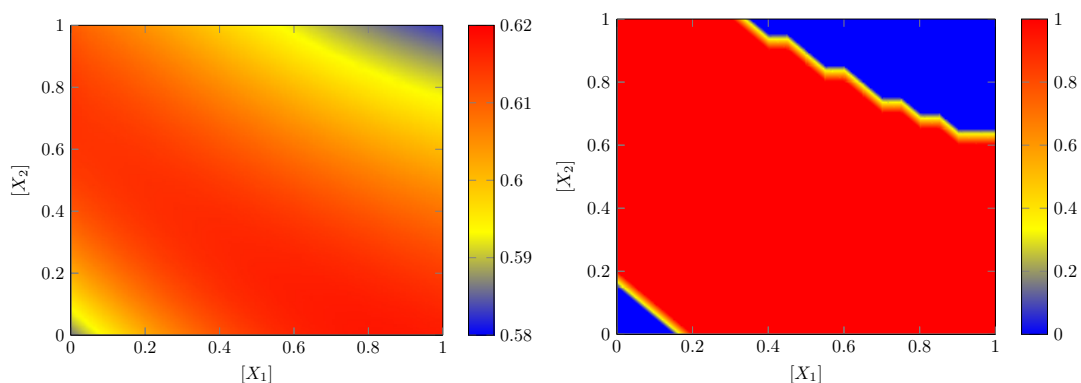
Figure 7.9: Response surface of an FCNN which learned XOR. Here input $[X]$ values of 1.0 correspond to TRUE, and 0.0 to FALSE, so the accuracy of the FCNN is defined only by its response at the corners of the plots. The plot on the left shows the FCNN's output value at each $([X_1], [X_2])$, while the plot on the right shows the same data thresholded by 0.6—the output values above the threshold correspond to TRUE (red region), and those below indicate FALSE (blue regions). Jagged lines in the right figure are an artifact of our sampling technique.

input $(0, 0)$ (i.e., when only S_{in} is present at injection), than to learn to output 1.

Unlike its building block, the AASP (Section 7.2.1), the FCNN in this context behaves as a binary perceptron. Thus, we compare its performance on the binary logic functions not with the AASP, but single chemical perceptrons: the WRP, the WRP, the SASP (Standard ASP) and an automatically thresholded version, the TASP (Thresholded ASP). As shown in Table 7.2, the FCNN is significantly more capable than any of these perceptrons.

Thus, the FCNN is the first chemical system to autonomously learn linearly insepa-rable functions. As shown by Minsky and Papert [110], single perceptrons cannot learn such functions because, as binary classifiers, they can only divide the input space along

a hyperplane. Like a multilayer perceptron, the FCNN does not have this constraint. To illustrate this, we mapped the response surface of an FCNN, which had successfully learned XOR, as shown in Figure 7.9.

7.6 DISCUSSION

We have presented a hierarchical, compartmentalized chemical system that is capable of autonomous learning. The FCNN is, to our knowledge, the first chemical system to learn a linearly inseparable function. It does so by a chemical analog of backpropagation. This is demonstrated in the classic example of the XOR binary function, which the FCNN learns perfectly in under 1,500 learning iterations, 100% of the time.

Each chemical neuron in the FCNN is a modified version of the AASP from our previous work. Neurons are distinguished from each other by their compartmentalization in nested cellular walls. This nesting constrains FCNNs to tree-like topologies, but allows modular design. Inter-neuron communication, facilitating feeding-forward and backpropagation, is mediated by selective permeation of signal species through the cell walls.

Each hidden neuron is chemically identical in terms of the species and reactions that occur within them. This means that the FCNN is easily extendable: once an FCNN with a simple topology has been implemented in wet chemistry, it will be possible to construct much larger networks than the minimal example explored here.

One technical complication in the FCNN design worth addressing is the manual injection of the feedforward signal-species S_F. Ideally, a chemical neural net could operate completely independently besides receiving input and a desired output signal. We are hopeful of developing mechanisms by which S_F could be generated periodically within the FCNN, perhaps using chemical oscillators such as the Lotka-Volterra predator-prey

system [96, 97].

Another direction for further research is optimizing the modules of the FCNN. We have developed a small family of single chemical perceptrons (Chapters 4 and 5). We have explored network architectures with many of these as building blocks, with performance presented in Appendix, Figure B.1. Though we have presented the best-performing architecture we have tested, it is possible that there are yet better components for the FCNN than our current hidden and output neurons.

Moreover, it would be intriguing to explore the FCNN model on larger networks, with either more hidden layers, more neurons per layer, or both. In related work Josh Moles [115] modeled a control system with an FCNN containing up to 4 inner compartments to tackle Santa Fe trail and John Muir trail [80] however instead of error backpropagation the weight concentrations were optimized by standard genetic algorithms. Larger FCNNs are expected to tackle more complex problems, and it is an interesting and open question how many layers and neurons per layer are necessary for the FCNN to solve certain tasks, as its behavior and performance will likely differ somewhat from classically implemented neural networks.

It bears repeating that the FCNN is not a chemical transcription of a neural network, but an analogy. As discussed with our results in Section 7.5, the FCNN we used here to solve XOR converged about as quickly as Rumelhart, Hinton, and Williams's first XOR-solving network [133]. We wonder if the FCNN generally converges faster or slower than classical neural nets. Our analog of input-weight integration, mathematically written in the language of ODEs, is highly nonlinear—as the building block of a backpropagated network, how does this compare to standard linear integration and sigmoidal activation functions?

The importance of this research is the hope of a reprogrammable chemical computer.

Like the formal multilayered perceptron, the FCNN has two modes: simply processing output when given an input, and learning via backpropagation of errors. A wet implementation of the FCNN, once trained to learn a task, could perform that task reliably as long as desired. The chemical computer could then, at any time, be retrained to perform any other task of which it is capable. We are hopeful that current work in synthesizing bilayer lipid membranes will develop a compartmentalized system with channels functionality equivalent to the FCNN's eventually opening profound possibilities for patient-embedded biochemical computers. There has already been significant research into medical uses of computational chemical networks. One recent result [162] presented a chemical logic circuit which tests for the presence of micro RNA molecules and the absence of others, effectively identifying a type of cancer cell called HeLa. The authors designed a bespoke chemical logic circuit for this purpose, but this is exactly the type of computation an FCNN excels at learning.

8

ANALYTIC BASIS OF CHEMICAL LEARNING

In previous chapters we explored several binary chemical perceptrons [21, 22], an analog chemical perceptron called the analog asymmetric signal perceptron [20], modeled delay lines [19, 114], and built a multi-compartment chemical feedforward network [27]. Even though these endeavours clearly demonstrated the feasibility of chemically implemented learning with teacher supervision for, e.g., smart drug delivery and chemical computing, we used formal neural networks just as a source of inspiration and we aimed to mimic their behavior qualitatively. Chemical reaction primitives differ from neural networks, so there is no simple one-to-one mapping between these two domains.

The complexity of differential equations, which describe the evolution of species concentrations, prevented us to gain a proper insight into the dynamics of our models and predict the behavior purely from their structure, i.e., without "running" the models using a numerical integration. Thus, a proof of our design choices and effectiveness of our learners were purely based on a statistical approach, extensive learning simulations. A mathematically rigorous analysis was, however, missing.

In this chapter we aim to fill this missing part and ask *why* chemical learning works. After flattening the catalytic reactions to mass-action kinetics, we succeeded to analyze the newly introduced linear cumulative input-weight integration and partially also the nonlinear cross-dependent input-weight integration, employed by the analog asymmetric

signal perceptron.

Instead of claiming that our chemical learners are "a metaphor" or qualitative coarse-grained implementation of formal perceptrons we show that the closed formulas we derived for the input-weight integration as well the weight update follow the mathematics of the formal linear perceptron. Our findings bridge adaptive chemical reaction networks and neural networks and open possibilities for applying rich findings of theory of neural networks to chemistry. For the nonlinear cross-dependent input-weight integration used in our previous work, we derive an approximative formula for an instance with a single input and weight. We show that cross-dependence prevents integrability for more than two inputs. This unwanted aspect also makes the analog asymmetric signal perceptron unscalable.

Moreover, we introduce the reactions implementing learning rate annealing, which benefits convergence and performance. A closed formula for the new annealed weight update corresponds closely to the classic delta rule known from machine learning. Because of cumulativity and independent processing of the input-weight branches, a new linear chemical perceptron learns 6 two-input linear and nonlinear functions 94 or 437 times better on average depending on the error metrics than the analog asymmetric signal perceptron. Once integrated with a delay line our new model is also more scalable and reaches a significantly smaller error of the $0.004 - 0.346$ RNMSE and the $0.02 - 4.83\%$ SAMP on benchmark time series.

Last but not least, we combine nonlinearity of the cross-dependent input-weight integration with cumulativity of a linear perceptron and derive a chemical sigmoid perceptron, which resembles its neural network counterpart. Having closed formulas rather than numerically integrated differential equations greatly reduces simulation costs.

8.1 KINETICS

As introduced in Section 2.3.1 the reaction rate of an ordinary reaction $aS_1 + bS_2 \rightarrow P$ with a rate constant $k \in \mathbb{R}^+$ is defined by mass-action law [49] as

$$r = \frac{d[P]}{dt} = -\frac{1}{a}\frac{d[S_1]}{dt} = -\frac{1}{b}\frac{d[S_2]}{dt} = k[S_1]^a[S_2]^b. \qquad (8.1)$$

In our previous models, we generated the differential equations of the catalytic reactions $S \xrightarrow{E} P$, where E is a catalyst, by Michaelis-Menten kinetics as

$$r = \frac{d[P]}{dt} = -\frac{d[S]}{dt} = \frac{k_{cat}[E][S]}{K_m + [S]}. \qquad (8.2)$$

Here we expand the catalytic reactions simply to $S + E \rightarrow P + E$ and replace the Michaelis-Menten with mass-action kinetics as

$$r = \frac{d[P]}{dt} = -\frac{d[S]}{dt} = k[E][S]. \qquad (8.3)$$

Recall that Michaelis-Menten kinetics are an approximation of the mass-action kinetics for two partial, associative and disassociative reactions, $E + S \rightleftharpoons$ and $ES \rightarrow E + P$, if the substrate S is in instantaneous equilibrium with the enzyme-substrate complex ES and the enzyme concentration $[E]$ is much smaller than the substrate concentration $[S]$. Using the mass-action variant is therefore more general and the resulting differential equations are easier to analyze.

8.2 INPUT-WEIGHT INTEGRATION

The input-weight integration is a key part of the chemical and neural network perceptrons. It produces the output from given inputs processed (integrated) through the weights. In our CRN implementations, the weights are catalysts of the input-output reactions.

In this section we present two variants for a chemical integration of the inputs and the weights: nonlinear cross-dependent and linear cumulative. The ODEs of the nonlinear cross-dependent input-weight integration employed by the analog asymmetric signal perceptron resists a rigorous analysis, however, the linear cumulative version allows to derive a closed formula that resembles that of a linear neural network perceptron. At the end of this section we combine the cumulativity of the new input-weight integration with a nonlinear activation function harvested from a single input cross-weight integration and derive the so-called *sigmoid chemical perceptron*.

8.2.1 Nonlinear Cross-Dependent Input-Weight Integration

The AASP presented in Chapter 5 consists of 17 species and 18 reactions and mimics a formal two-input analog perceptron [132]. The AASP integrates the inputs and the weights in a cross-dependent fashion. During a nonlinear input-weight integration (Figure 5.1(a)), each weight W_i catalyzes a transformation of the input X_i to the output Y and races with the annihilation of its input and the output $X_i + Y \rightarrow \lambda$. For clarity, we relabel the input signal S_{in} to a zero$^{\text{th}}$ input X_0.

Recall that if the concentration of a weight W_i is high the input-output transformation proceeds rapidly and the annihilation fails to interfere, since both the input X_i and the output Y are simultaneously present only for a short period of time. In the opposite case when the weight concentration is close to zero, its branch could consume more from

the output than it contributes to. Therefore, a weight species coupled with its annihilatory reaction can act effectively as a catalyst and an inhibitor. Since the global output Y is shared, each weight races besides the annihilation in its own branch also with other input-weight branches. A low weight concentration could impose a negative pressure on a different weight branch, and therefore this type of input-weight integration allows to represent both addition and subtraction in a nonlinear weight cross-dependent form. Note that besides the output Y each input transforms also to the species X_i^L (by the reaction $X_i \xrightarrow{W_i} Y + X_i^L$), which keeps a record of how much each input contributed to the output and is used later for the weight adaptation.

As we stated initially, we opt for a rigorous analysis of the chemical input-weight integration and weight update. In this section we show, however, that the catalytic reactions of the cross-weight input-weight integration resists deriving a closed formula for the output from the underlying ordinary differential equations. That is due to the system's inherent complexity, which arises from cluttering all input-weight branches. Also, the cross-weight dependency results in poor scalability, as we demonstrate in Section 8.5. On the other hand, we approximate the output formula by lower and upper bounds, for the case of a single input and weight, which provides a new insight into the system's behavior.

Analysis

Here we analyze a cross-dependent input-weight integration with the catalytic reactions following mass-action kinetics. We proceed with general rate constants, which makes the outcomes applicable for the AASP as well.

We start with the case of a single input X and the corresponding weight W. The reactions and rates follow.

$$X \xrightarrow{W} Y + X^L \quad \Big| \quad k_1[X][W]$$

$$X + Y \rightarrow \lambda \quad \Big| \quad k_2[X][Y]$$

(8.4)
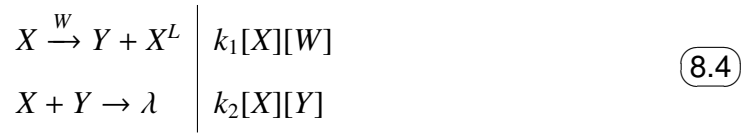
We assume each rate constant is positive, i.e., each reaction actually proceeds (exists). In the remainder of this chapter, we will use lower case letters to represent the species concentrations, i.e., $x = [X]$, $x^L = [X^L]$, $y = [Y]$, and $w = [W]$. As mentioned before X^L is a contribution-keeping species, which plays an important role only in a weight update, but during a input-weight integration is extraneous.

The corresponding system of ODEs is

$$\frac{dx}{dt} = -k_1 xw - k_2 xy = x(-k_1 w - k_2 y)$$

$$\frac{dy}{dt} = k_1 xw - k_2 xy = x(k_1 w - k_2 y)$$

$$\frac{dx^L}{dt} = k_1 xw.$$

(8.5)
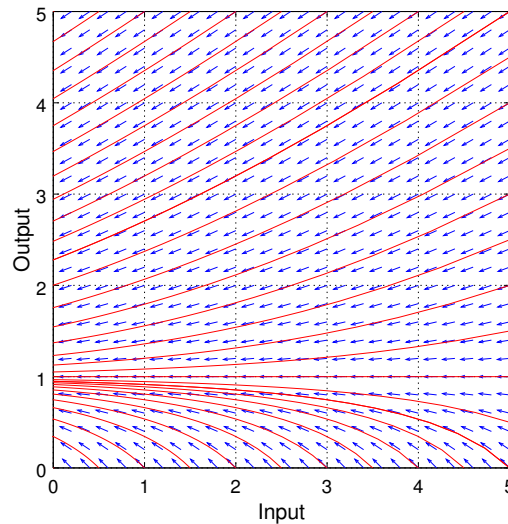
Figure 8.1: The input-output vector map of the nonlinear cross-dependent input-weight integration with a single input (weight) and $k_1 = k_2 = w = 1$. Note that the output decreases above the threshold concentration $C = \frac{k_1 w}{k_2} = 1$ and increases below.

It follows that $\frac{dy}{dt} = 0$ *iff* $x = 0$ (no input left) or $y = C$, where $C = \frac{k_1 w}{k_2}$. Therefore, if $y > C$ the output concentration decreases ($\frac{dy}{dt} < 0$) and if $y < C$, the output concentration increases ($\frac{dy}{dt} > 0$) as shown in Figure 8.1.

By substituting $x = (-k_1 w - k_2 y)^{-1} \frac{dx}{dt}$ from the first equation into the second, we integrate the $\dot{y}$-equation directly

$$
\begin{aligned}
\frac{dy}{dt} &= x(k_1 w - k_2 y) \\
\frac{dy}{dt} &= \frac{k_1 w - k_2 y}{-k_1 w - k_2 y} \frac{dx}{dt} \\
\frac{-k_1 w - k_2 y}{k_1 w - k_2 y} dy &= dx \\
(1 + \frac{-2k_1 w}{k_1 w - k_2 y}) dy &= dx \\
\int_{y_0}^{y_t} (1 + \frac{-2k_1 w}{k_1 w - k_2 y}) dy &= \int_{x_0}^{x_t} dx
\end{aligned}
\tag{8.6}
$$

$$
y_t - y_0 + 2C(\ln(k_1 w - k_2 y_t) - \ln(k_1 w - k_2 y_0)) = x_t - x_0.
$$

For all our species (especially the output y_t) we are interested primarily in the final concentration. To conceptualize a final state, in which the system reaches equilibrium (all substrate is consumed), we introduce a special symbol $\perp$ that denotes the eventual equilibrium time. Naturally, from the analytic perspective, the time $t = \perp$ is $t \to \infty$, but in practical applications, where we alter input-weight integration and weight update phases many times sequentially to achieve a desired behavior, $t = \perp$ for a single output production is set to a finite, sufficiently large number such that most of the input is consumed and the state of the system could be factually perceived as a fixed point. For the simulations (Section 8.5) we specify this time period for both input-weight integration and weight update more precisely. In all analytical considerations of the eventual behavior we will, however, use $t = \perp$ for $t \to \infty$ interchangeably.

(a) Output approximation (b) Output approximation (input on a log scale)
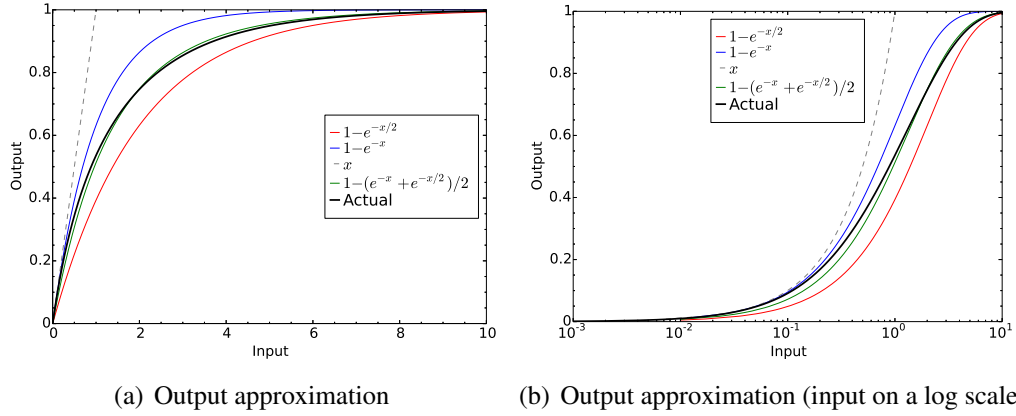
Figure 8.2: The input-output relation of the nonlinear cross-weight input-weight integration with a single input compared to the lower bound $1 - e^{-\frac{x}{2w}}$, the upper bound $1 - e^{\frac{-x}{w}}$, the mean approximation $w\left(1 - \frac{e^{\frac{x}{2w}} + e^{\frac{x}{w}}}{2}\right)$, and a linear function. The rate constants and the weight concentration were set to 1 ($k_1 = k_2 = w = 1$).

Now, since eventually all the input X is consumed and initially no output is present, we set $y_0 = 0$ and $x_\perp = 0$, which implies

$$y_\perp + 2C(\ln(k_1 w - k_2 y_\perp) - \ln(k_1 w)) = -x_0$$

$$y_\perp + 2C \ln(1 - C^{-1} y_\perp) = -x_0.$$

(8.7)

The output cannot exceed $C = \frac{k_1 w}{k_2}$ regardless of the input. To simplify the formula we denote $y = y_\perp$ as the final output and $x = x_0$ as a given (initial) input. Because both y and $\ln(f(y))$ appear on the left side, the expression cannot be stated in a closed form. Despite this difficulty we characterize the behavior of a single input nonlinear input-weight integration by bounding and approximating the output formula. First, by using the Maclaurin expansion, we express $\ln(1 - q)$ as $- \sum_i \frac{q^i}{i}$, which gives us

$$\ln(1 - C^{-1} y) = - \sum_i \frac{(C^{-1} y)^i}{i} \leq -C^{-1} y,$$

(8.8)

and so the upper bound is

$$C \ln(1 - C^{-1}y) - x \leq -y - x = 2C \ln(1 - C^{-1}y)$$

$$C \ln(1 - C^{-1}y) \geq -x$$

$$1 - C^{-1}y \geq e^{\frac{-x}{C}}$$

$$y \leq C(1 - e^{\frac{-x}{C}}).$$

(8.9)

For the lower bound we simply drop the $y+$ term on the left side of Equation 8.7 and derive a similar inequality

$$2C \ln(1 - C^{-1}y) \leq y + 2C \ln(1 - C^{-1}y) = -x$$

$$\ln(1 - C^{-1}y) \leq \frac{-x}{2C}$$

$$y \geq C(1 - e^{\frac{-x}{2C}}).$$

(8.10)

By combining both results we bound y as

$$C(1 - e^{\frac{-x}{2C}}) \leq y \leq C(1 - e^{\frac{-x}{C}}).$$

(8.11)

It is noteworthy that the actual y is closely approximated by the upper bound $C(1 - e^{\frac{-x}{C}})$ for $y \approx 0$, where both y and $C(1 - e^{\frac{-x}{C}})$ behave as a linear function $y = x$ (Figure 8.2). The linear nature of y when close to zero can be seen also through the Maclaurin expansion, since $\ln(1 - C^{-1}y) = -C^{-1}y - O(y^2) \approx -C^{-1}y$ for $y \approx 0$, and so, $-x = y + 2C \ln(1 - C^{-1}y) \approx y - 2y = -y$. For the opposite case, where y is close to the saturation point $y \approx C$, $-2C \ln(1 - C^{-1}y) \gg y$, and $y \approx C(1 - e^{\frac{-x}{2C}})$ (the lower bound).

For convenience we set $k_1 = k_2 = 1$ and bound y as

$$w(1 - e^{\frac{-x}{2w}}) \leq y \leq w(1 - e^{\frac{-x}{w}}).$$

(8.12)

Alternatively we can approximate y by a simple average of these two bounds

$$y \approx w\left(1 - \frac{e^{\frac{-x}{w}} + e^{\frac{-x}{2w}}}{2}\right). \tag{8.13}$$

Even though we succeeded in approximating the input-output relation of the nonlinear cross-dependent input-weight integration with a single input, the underlying ODEs become significantly more complicated for two and more inputs, and cannot be explicitly integrated or approximated sufficiently. The general differential output formula for n inputs is

$$dy = \sum_i \left(1 + \frac{2k_1 w_i}{-k_1 w_i - k_2 y}\right)dx_i. \tag{8.14}$$

Equation 8.14 underlines the fact that the weight contributions are interlinked and cannot be separated. Since the weights race with each other, the contribution of each weight does not depend solely on its own input, but also on the inputs of the remaining weights.

Recall that the purpose of a species X_i^L is to track the contribution of the input-weight branch W_i. During the weight adaptation each weight is supposed to be adjusted "proportionally" to its own contributing record X_i. However, as the number of inputs grow, the information stored in X_i^L becomes less reliable and misleads the weight adaptation process, since a certain weight could have had a different effect on the output if other inputs (and weights) were different. This results in an unfair change of the weights.

In Section 8.5 we illustrate that the nonlinear cross-dependent input-weight integration does not scale well. For a large number of inputs, the performance drops and the weights become highly correlated. In fact, the weights start acting as a single weight and converge towards the mean of their input stream.

8.2.2 Linear Cumulative Input-Weight Integration

As we showed, the nonlinear cross-dependent input-weight integration is not properly analyzable and its scalability is poor. The reason behind that is a parallel annihilation of the inputs with the global output.

A naive way to tackle that is to replace the input-output annihilations with input decays as shown in Figure 8.3(a). This makes the output Y a sink (a terminal species)—it can act only as a product, but not as a reactant, i.e., reaction arrows can only enter Y. A combination of the partial outputs thus becomes strictly additive. Despite processing each input-weight branch independently, we could not represent the situations where some of the formal weights need to be "negative." Specifically, the smallest achievable contribution of a weight species W_i is zero (for $[W_i] = 0$), but could never be negative.

(a) Strictly additive   (b) Cumulative
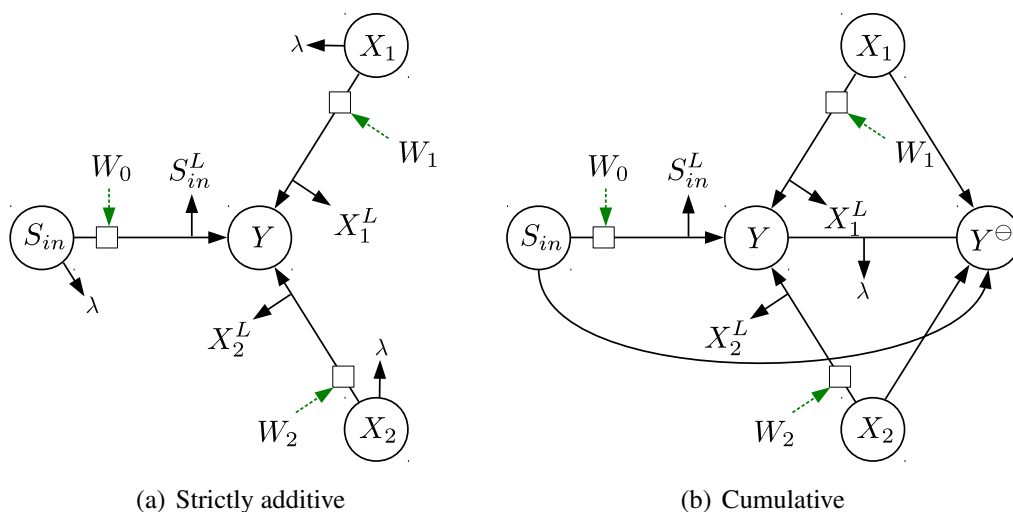
Figure 8.3: The reactions performing: a) a strictly additive input-weight integration of two inputs, where each weight races with decay of its input X_i, b) a linear cumulative input-weight integration of two inputs, where each weight races with a transformation of its input to the negative output $Y^\ominus$. The positive and negative outputs combine through the annihilation $Y + Y^\ominus \rightarrow \lambda$. In both cases three species S_{in}^L, X_1^L, and X_2^L represent the contributions of the inputs S_{in}, X_1, and X_2 with associated weights in the output Y. Nodes represent species, solid lines are reactions, dashed lines are catalysts, and λ stands for no or inert species.

Formally, we aim to achieve "cumulativity" (and independence) of the input-weight branches such that the output $y = \sum_i f(x_i, w_i)$, where $f(x_i, w_i)$, could hold both positive and negative numbers. To keep the negative partial outcomes, we avoid decaying the inputs and rather transform them to a new species $Y^\ominus$. Each weight W_i therefore races with a constant negative transformation $X_i \rightarrow Y^\ominus$. To create the global output, we let Y and $Y^\ominus$ annihilate. Although the outputs Y and $Y^\ominus$ react and are not terminal species *per se*, perceived together, they form a sink of the system.

Later we will use the sink property of Y and $Y^\ominus$ (Section 8.2.2) and show that the input-weight integration is indeed cumulative and performs both addition and subtraction. In fact, the simplicity of the reactions allows for a direct integration of the ODEs and we succeed to derive a closed formula for the global output as $y = \sum(1 - \frac{2}{w_i+1})x_i$, which resembles that of a linear neural network perceptron. The behavior of the linear chemical perceptron is more predictable, it is easier to control, and scales and performs better than the AASP, which utilizes the nonlinear cross-dependent input-weight integration.

Analysis

We start with the case of a single input X and the corresponding weight W. The reactions and rates follow.

$$
\begin{array}{ll}
X \xrightarrow{W} Y + X^L & k_1[X][W] \\
X \rightarrow Y^\ominus + X^L & k_2[X] \\
Y + Y^\ominus \rightarrow \lambda & k_3[Y][Y^\ominus]
\end{array}
\qquad (8.15)
$$
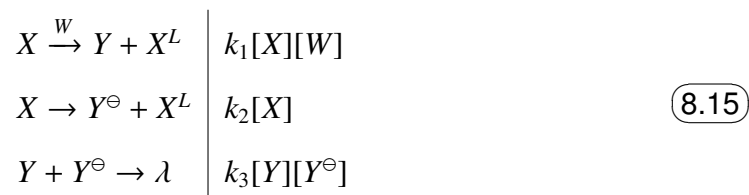
Let $x = [X]$, $y = [Y]$, $y^\ominus = [Y^\ominus]$, $x^L = [X^L]$, and $w = [W]$. The corresponding system of ODEs is

$$\frac{dx}{dt} = -k_1 xw - k_2 x$$

$$\frac{dy}{dt} = k_1 xw - k_3 yy^\ominus$$

$$\frac{dy^\ominus}{dt} = k_2 x - k_3 yy^\ominus \qquad (8.16)$$

$$\frac{dx^L}{dt} = k_1 xw + k_2 x$$

We can integrate the equation $\dot{x}$ directly as any first-order reaction (setting $C = k_1 w + k_2$)

$$\frac{dx}{dt} = -Cx$$

$$\frac{1}{x} dx = -C dt$$

$$\int_{x_0}^{x} \frac{1}{x_t} dx = \int_{0}^{t} -C dt \qquad (8.17)$$

$$\ln x - \ln x_0 = -Ct$$

Hence, the concentration of X at time t is

$$x_t = x_0 \, e^{-Ct}. \qquad (8.18)$$

Now let $y^{\ominus\prime}_t$ be the concentration $y^\ominus_t$ as it would be without the annihilation $Y + Y^\ominus \rightarrow \lambda$, which would consume overall y^a_t of Y and $Y^\ominus$, i.e., $y^{\ominus\prime}_t = y^\ominus_t + y^a_t$ and $\frac{dy^{\ominus\prime}}{dt} = k_2 x$. When we substitute $x = -\frac{1}{C}\frac{dx}{dt}$ from the first equation, we obtain

$$\frac{dy^{\ominus\prime}}{dt} = -\frac{k_2}{C}\frac{dx}{dt}$$

$$\int_{y^\ominus_0}^{y^\ominus_t} dy^{\ominus\prime} = \int_{x_0}^{x_t} -\frac{k_2}{C} dx \qquad (8.19)$$

$$y^{\ominus\prime}_t = -\frac{k_2}{C}(x_t - x_0) + y^{\ominus\prime}_0.$$

Similarly, $y'_t = y_t + y^a_t$ and $y'_t = -\frac{k_1 w}{C}(x_t - x_0) + y'_0$. Since the system is closed and all reactions have the same number of reactants and products (excluding the annihilation $Y + Y^\ominus \to \lambda$), we know that the total concentration is preserved, i.e.,

$$y'_t + y^{\ominus\prime}_t + x_t = y_0 + y^\ominus_0 + x_0. \tag{8.20}$$

Now, let us assume the usual initial state of $y_0 = 0$, and $y^\ominus_0 = 0$. Then the concentration y'_t equals

$$y'_t = x_0 - y^{\ominus\prime}_t - x_t. \tag{8.21}$$

Because the annihilatory constant is $k_3 > 0$, we know that eventually Y and $Y^\ominus$ fully annihilate, leaving a leftover of either Y or $Y^\ominus$. Using the symbol $\perp$ for an asymptotic time, eventually $y^a_\perp = y^{\ominus}_\perp$ or $y^a_\perp = y'_\perp$. Trivially, the input is only consumed, hence $x_\perp = 0$.

Suppose $y'_\perp \geq y^{\ominus\prime}_\perp$, which holds for $k_1 w \geq k_2$. Then $y^a_\perp = y^{\ominus\prime}_\perp$, and $y_\perp = y'_\perp - y^a_\perp \geq 0$. Further,

$$
\begin{aligned}
y_\perp &= y'_\perp - y^a_\perp \\
y_\perp &= x_0 - y^{\ominus\prime}_\perp - x_\perp - y^a_\perp && \text{(Equation 8.21)} \\
y_\perp &= x_0 - y^{\ominus\prime}_\perp - x_\perp - y^{\ominus\prime}_\perp && (y^a_\perp = y^{\ominus\prime}_\perp \text{ by the assumption } y'_\perp \geq y^{\ominus\prime}_\perp) \\
y_\perp &= x_0 + 2\frac{k_2}{C}(x_\perp - x_0) - x_\perp && \text{(Equation 8.19)} \\
y_\perp &= (1 - \frac{2k_2}{C})x_0 \\
y_\perp &= (1 - \frac{2k_2}{k_1 w + k_2})x_0.
\end{aligned}
\tag{8.22}
$$

Since the output species Y and $Y^\ominus$ form a sink of all the input-weight branches, their

contributions are cumulative and independent from each other. If we denote the final output $y_\perp$ simply as the total output concentration for n inputs $x_0, \ldots, x_n$ with weights $w_0, \ldots, w_n$,

$$y = \sum_i \left(1 - \frac{2k_2}{k_1 w_i + k_2}\right) x_i. \tag{8.23}$$

Equation 8.23 shows that the cumulative input-weight integration is linear from the input perspective, although the effect of a weight w_i on the input x_i's factor is nonlinear and equals $1 - \frac{2k_2}{k_1 w_i + k_2}$. Similarly to the sigmoid function, this factor ranges between -1 and 1 and reaches zero for $w_i = \frac{k_1}{k_2}$. For convenience we center the formal zero-weight value to $w = 1$ by setting $k_1 = k_2 = 1$, finally simplifying the input-output relation to

$$y = \sum_i \left(1 - \frac{2}{w_i + 1}\right) x_i. \tag{8.24}$$

By introducing a formal chemical weight $w'_i = 1 - \frac{2}{w_i+1}$, the formula changes to

$$y = \sum_i w'_i x_i. \tag{8.25}$$

A formal weight w'_i holds a negative value for w_i between 0 and 1 and positive for $w_i > 1$. The formula we derived is very similar to that of a classic linear perceptron, where the input-weight integration is calculated as a dot product

$$y = \sum_i w_i x_i. \tag{8.26}$$

The only difference here is that the values of w'_i could range between -1 and 1, as opposed to a formal linear perceptron where weights are unbounded and can hold an arbitrary real value. It is important to realize that the constraint on chemical weights

is due to the preservation of matter. Specifically, the only substrate each weight could process is its input X_i, therefore, the input X_i can by either fully subtracted or fully added to the output (or anything between) but could never become for instance $2[X_i]$. Thus, the total output is bounded by $\sum_i [X_i]$.

8.2.3 Sigmoid Chemical Perceptron

An evident benefit of the linear input-weight integration with the output formula $y = \sum w_i' x_i$, is cumulativity and independence of the weights. In the theory of neural networks, an input-weight integration is followed by applying an activation function f. For more complicated learning scenarios it is beneficial to choose f as a nonlinear function (tanh, logistic function, etc.). To incorporate a nonlinear activation function into the cumulative input-weight integration, we harness the nonlinearity of a single input cross-dependent input-weight integration.

Recall that the input-output relation of the cross-dependent input-weight integration has a sigmoid shape similar to *tanh*, and could be approximated by $w\left(1 - \frac{e^{\frac{-x}{w}} + e^{\frac{-x}{2w}}}{2}\right)$. We adapt the reactions of the linear cumulative input-weight integration such that instead of Y and $Y^\ominus$ it produces the intermediate species Z and $Z^\ominus$, which represent a signed dot product. More precisely,

$$
\begin{array}{ll}
X_i \xrightarrow{W_i} Z + X_i^L & k_1[X_i][W_i] \\
X_i \rightarrow Z^\ominus + X_i^L & k_2[X_i] \\
Z + Z^\ominus \rightarrow \lambda & k_3[Z][Z^\ominus].
\end{array}
\tag{8.27}
$$
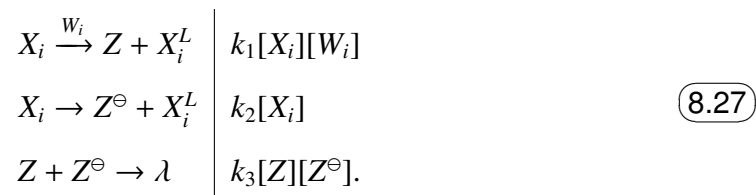
On the top of that we add the reactions for a sigmoid processing of the species Z, since we care only about the positive output. Here we drop the weight W from the original cross-dependent integration (we replace a catalytic reaction $Z \xrightarrow{W} Y$ with $Z \rightarrow Y$) and

obtain the reactions

$$
\begin{array}{c|c}
Z \to Y & q_1[Z] \\
Z + Y \to \lambda & q_2[Z][Y].
\end{array}
\tag{8.28}
$$
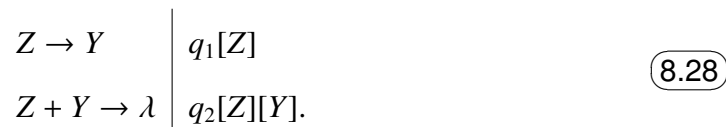
Similarly to the linear cumulative input-weight integration, we set $q_1 = q_2 = 1$. To make the sigmoid activation function to work in a modular fashion, we need to assume that the input-weight integration part finishes "sooner" than the sigmoid part. We achieve this behavior by assuming the sigmoid processing reactions are slower, i.e., $q_1 = q_2 \ll k_1 = k_2$. Also, to minimize an error, the annihilation $Z + Z^\ominus \to \lambda$ is assumed to be instant $k_3 \gg q_1$. Alternatively, we could introduce a special trigger signal S_A to guard the output reaction of the sigmoid activation function, i.e., $Z \xrightarrow{S_A} Y$. In this case the rate constants $q_1 = q_2$ would not need to be smaller than k_1, k_2, and k_3.

Finally, the sigmoid chemical perceptron can be formalized as

$$
y = f(z), z = \sum_i (1 - \frac{2}{w_i + 1})x_i,
\tag{8.29}
$$

where $f \approx 1 - \frac{e^{-x} + e^{\frac{-x}{2}}}{2}$.

8.3 LEARNING AND WEIGHT UPDATE

After the output is produced through a linear or nonlinear input-weight integration, we train a chemical perceptron by providing a desired output. A discrepancy between the actual and desired output propagates through the chemical perceptron and results in an adaptation of the weight concentrations.

In this section we present two ways for updating the weights. In the first scenario, which we employed before for training the AASP (chapter 5), the total amount by which

all the weights are updated directly corresponds to the difference between the actual and desired output. The second, newly introduced approach amplifies or reduces the output difference by a factor, which we anneal over time. In the machine learning community this is commonly known as a learning rate. Annealing a learning rate over time benefits weight convergence and overall performance.

Note that the weight update is modular and could be combined with arbitrary input-weight integration and activation function. The only linkage to the learning phase from a preceding input-weight integration is the weight contribution species X_i^L, whose values (and meaning) depend on the integration type.
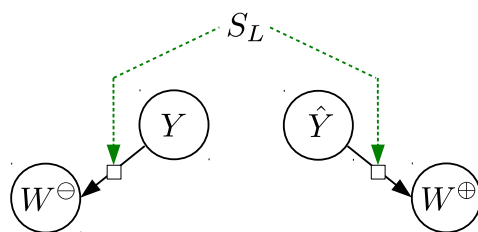
Figure 8.4: The reactions responsible for a direct production of weight changers $W^\ominus$ and $W^\oplus$ from the output and desired output species Y and $Y^\ominus$ triggered by a learning signal S_L.

8.3.1 Direct Weight Update

Our chemical implementation of the classical supervised learning [130] is triggered by an injection of the target output $\hat{Y}$. The target output is provided after the input species, injected at the beginning of the input-weight integration phase, transforms to the output Y.

Intuitively, a large output Y compared to the desired output $\hat{Y}$ implies that the weights need to be decreased, hence we produce a negative weight-changer $W^\ominus$ from Y. In the opposite case, $\hat{Y}$ is transformed to a positive weight-changer $W^\oplus$ to increase the weights. The species $W^\oplus$ and $W^\ominus$ represent a total concentration by which all the weights get

(a) Positive weight adaptation (b) Negative weight adaptation
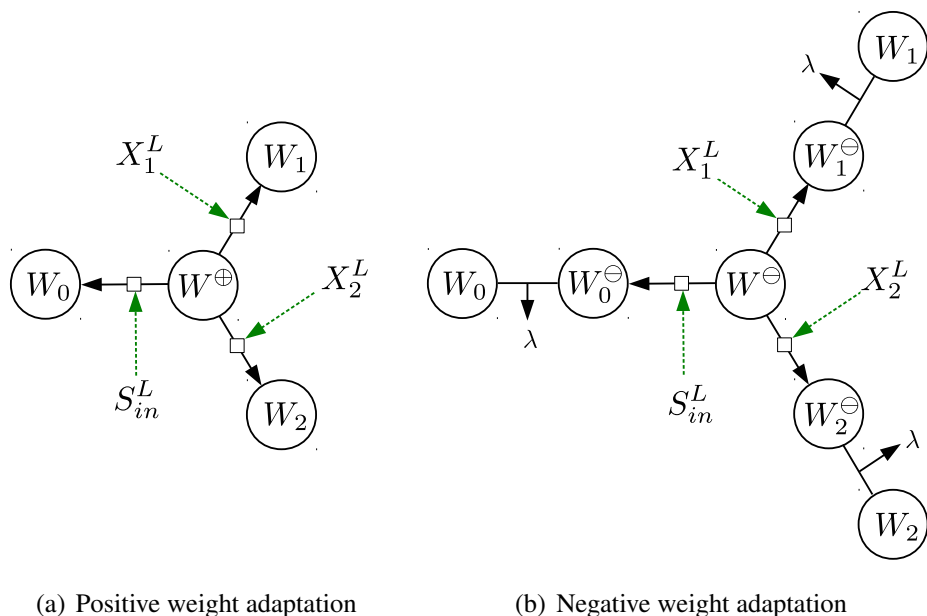
Figure 8.5: The reactions responsible for positive and negative weight adaptation.

updated. Also, since the output species Y is continuously produced during an input-weight integration, we need to guard the reaction $Y \rightarrow W^{\ominus}$ by a learning signal S_L that is injected with the target output and removed afterwards (Figure 8.4). For consistency, we let S_L trigger also the reaction $\hat{Y} \rightarrow W^{\ominus}$, although the species $\hat{Y}$ is injected instantaneously only when needed so even without an explicit reaction trigger (a catalyst), its transformation would work correctly.

Having a production of the total weight-changers $W^{\oplus}$ and $W^{\ominus}$ covered, we need to distribute them among the weights. For the positive adaptation, $W^{\oplus}$ is split by concurrent catalytic reactions $W^{\oplus} \rightarrow W_i$ among the weights proportionally to their input-weight contributions X_i^L (Figure 8.5(a)). Similarly, the negative adaptation splits $W^{\ominus}$ to intermediates $W_i^{\ominus}$, which annihilate with the weights (Figure 8.5(b)). It is critical to mention that both the positive and negative weight updates occur simultaneously and drive the weights in opposing directions. Eventually, through an annihilation, $W_i + W_i^{\ominus} \rightarrow \lambda$. The net in-

crease or decrease of a weight W_i is decided by a difference between the concentration of the total weight-changers $[W^\oplus] - [W^\ominus]$, which equals a difference between $[\hat{Y}]$ and $[Y]$. We elaborate this argument more thoroughly in Section 8.3.1 and show that an initial annihilation $Y + \hat{Y} \rightarrow \lambda$ would be redundant and yields the same eventual weight change.

Note that X_i^L is an auxiliary species that represents a contribution of the input X_i with the associated weight W_i in the output Y. After each learning iteration, the input-weight contributions species are flushed. The specific value (concentration) of X_i^L depends on the type of input-weight integration that produced it.

Analysis

The reactions and rates of a direct weight update follow.

$$
\begin{array}{ll|l}
W^\oplus \xrightarrow{X_i^L} W_i & k_1[W^\oplus][X_i^L] & \forall i \in \{0, \dots, n\} \\[4pt]
W^\ominus \xrightarrow{X_i^L} W_i^\ominus & k_1[W^\ominus][X_i^L] & \forall i \in \{0, \dots, n\} \\[4pt]
W_i + W_i^\ominus \rightarrow \lambda & k_2[W_i][W_i^\ominus] & \forall i \in \{0, \dots, n\} \\[4pt]
\hat{Y} \rightarrow W^\oplus & k_3[\hat{Y}] & \\[4pt]
Y \rightarrow W^\ominus & k_3[Y] &
\end{array}
\tag{8.30}
$$
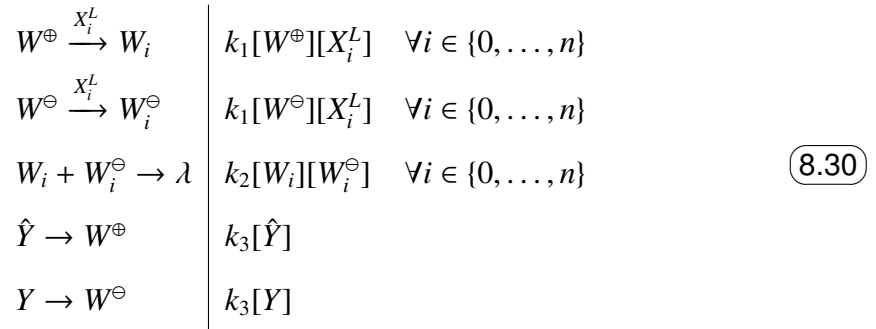
Let $x_i^L = [X_i^L]$, $y = [Y]$, $\hat{y} = [\hat{Y}]$, $w^\oplus = [W^\oplus]$, $w^\ominus = [W^\ominus]$, $w_i = [W_i]$, and $w_i^\ominus = [W_i^\ominus]$. First, suppose the weight-changer concentrations $w^\oplus$ and $w^\ominus$ are given, i.e., we ignore the last two reactions. Also, similarly to Section 8.2.2, we omit the annihilation $W_i + W_i^\ominus \rightarrow \lambda$ and set $w_i' = w_i + w_i^a$ and $w_i^{\ominus\prime} = w_i^\ominus + w_i^a$. Then the corresponding system of ODEs is

$$\frac{dw^{\oplus}}{dt} = -k_1 \sum_i x_i^L w^{\oplus}$$

$$\frac{dw^{\ominus}}{dt} = -k_1 \sum_i x_i^L w^{\ominus}$$

$$\frac{dw'_i}{dt} = k_1 x_i^L w^{\oplus} \qquad \forall i \in \{0, \ldots, n\}$$

$$\frac{dw^{\ominus'}_i}{dt} = k_1 x_i^L w^{\ominus} \qquad \forall i \in \{0, \ldots, n\}.$$

(8.31)

The concentration of the total weight-changers $w^{\oplus}$ or $w^{\ominus}$ is divided among the weights proportionally to their input concentrations. More precisely, we derive the formula for w'_{it} as follows

$$\frac{dw'_i}{dt} = k_1 x_i^L w^{\oplus}$$

$$dw'_i = -\frac{k_1 x_i^L}{k_1 \sum_j x_j^L} dw^{\oplus}$$

$$w'_{it} - w'_{i0} = \frac{x_i^L}{\sum_j x_j^L}(w_0^{\oplus} - w_t^{\oplus})$$

$$w'_{it} = \frac{x_i^L}{\sum_j x_j^L} w_t^{\oplus} + w_{i0} \qquad (w_{i0} = w'_{i0}, w_0^{\oplus} = 0).$$

(8.32)

Similarly, $w^{\ominus'}_i = \frac{x_i}{\sum_j x_j} w_t^{\ominus}$. Since the equations for w'_{it} and $w^{\ominus'}_{it}$ are linear with regards to $w_t^{\oplus}$ and $w_t^{\ominus}$, and the originally omitted reactions $\hat{Y} \to W^{\oplus}$ and $Y \to W^{\ominus}$ supply the species $W^{\oplus}$ and $W^{\ominus}$ uni-directionally, the total concentration $w^{\oplus}$ and $w^{\ominus}$ produced by those two reactions can be incorporated directly to w'_{it} and $w^{\ominus'}_{it}$. Regardless of k_3, eventually all the target output $\hat{Y}$ and the output Y transforms to $W^{\oplus}$ and $W^{\ominus}$ respectively. Hence

$$w'_{i\perp} = \frac{x_i^L}{\sum_j x_j^L} \hat{y}_0 + w_{i0}$$

$$w^{\ominus'}_{i\perp} = \frac{x_i^L}{\sum_j x_j^L} y_0.$$

(8.33)

To finalize the formula, we plug in the annihilatory reactions $W_i + W_i^{\ominus} \to \lambda$ and write

$$\Delta w_{i\perp} = \frac{x_i^L}{\sum_j x_j^L}(\hat{y}_0 - y_0). \tag{8.34}$$

If we denote a final $\Delta w_{i\perp}$ simply as Δw_i and a given output and desired output concentration as y and $\hat{y}$, the chemical weight update formula becomes

$$\Delta w_i = \frac{x_i^L}{\sum_j x_j^L}(\hat{y} - y). \tag{8.35}$$

Since a concentration is never negative, a new weight value $w_i = \Delta w_i + w_i$ is truncated at zero.

Note that because each weight depends on $\hat{y} - y$, a comparison of the output and the target output by the annihilation $Y + \hat{Y} \to \lambda$ (used by the analog asymmetric signal perceptron) yields the same (eventual) result and is therefore redundant. To prove that formally, suppose $Y + \hat{Y} \to \lambda$ is present. As we did before, let y_t^a be the amount of Y and $Y^{\ominus}$ consumed by the annihilation untill time t. Then $y_t' = y_t - y_t^a$ and $\hat{y}_t' = \hat{y}_t - y_t^a$, and eventually the total concentration of Y and $\hat{Y}$ provided for the weight adaptation would be $y_0 - y_{\perp}^a$ and $\hat{y}_0 - y_{\perp}^a$, and so $\Delta w_{i\perp} = \frac{x_i^L}{\sum_j x_j^L}(\hat{y}_0 - y_{\perp}^a - (y_0 - y_{\perp}^a))$, which equals Equation 8.34 and after relabeling also Equation 8.35.

We observe that the sum of all input contributions (a normalization factor) in Equation 8.35 statistically approaches a constant as the number of inputs grow, and so could be perceived as a (constant) learning rate. To avoid the normalization, we would need to replace the reactions $\hat{Y} \to W^{\oplus}$ and $Y \to W^{\ominus}$ with the reactions $\hat{Y} \to \hat{Y}_0 + \ldots + \hat{Y}_n$ and $Y \to Y_0 + \ldots + Y_n$ that clone $\hat{Y}$ and Y and allow each weight update to proceed independently. These reactions are, however, unscalable since each new weight would require changing the right side of the reactions and the large (potentially unbounded)

number of products would make any wet chemical implementation highly unpractical.

8.3.2 Annealed Weight Update

In a direct weight update, the error $|y - \hat{y}|$ translates to $\sum_i \Delta w_i$, which is split among the weights. Since the characteristics of a desired input-output function is unknown to a chemical perceptron, and in general could be arbitrary, the magnitude of a weight change does not necessarily correspond to that of the output. That could lead to the target over/under shooting and poor convergence.

This is a classic control problem of control theory and machine learning. A solution is to introduce a learning rate (an amplification factor) and anneal it over time. Initially the weights could explore the input-output surface in a wide range and over time, as the learning rate decreases, so does the amplitude of weights.

In our chemical learning implementation we incorporate a learning rate into the reactions of the direct weight update presented in Section 8.3.1. Instead of directly producing the total weight-changers $W^\oplus$ and $W^\ominus$ from $\hat{Y}$ and Y, we add an "error layer" represented by the positive and negative error species $E^\oplus$ and $E^\ominus$. These catalyze the production of $W^\oplus$
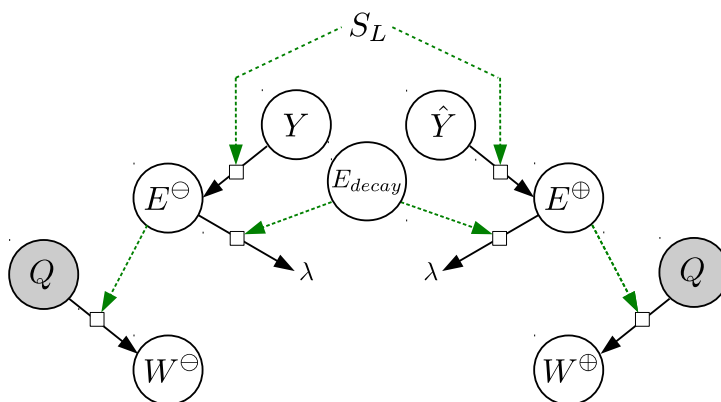
Figure 8.6: The reactions responsible for the annealed weight adaptation.

and $W^\ominus$ from an external (abundant) source Q, kept at a constant concentration $[Q] = 1$, as shown in Figure 8.6. A time window in which the error species $E^\oplus$ and $E^\ominus$ operate is tuned by the species E_{decay}, which catalyzes their decay.

The larger the time window, the larger the amount of Q transformed to $W^\oplus$ and $W\ominus$, and the more the original difference of $[Y]$ and $[\hat{Y}]$ (used for the weight adaptation) amplifies. Because $[E_{decay}]$ shrinks the error, it works essentially as an inverse of a learning rate. Instead of reducing $[E_{decay}]$, we increase it over time to emulate the effect of formal learning rate annealing, as done in neural networks.

Analysis

The reactions and rates of the annealed direct weight update are as following.

$$
\begin{array}{ll|l}
W^\oplus \xrightarrow{X_i^L} W_i & & k_1[W^\oplus][X_i^L] \\[4pt]
W^\ominus \xrightarrow{X_i^L} W_i^\ominus & & k_1[W^\ominus][X_i^L] \\[4pt]
W_i + W_i^\ominus \to \lambda & & k_2[W_i][W_i^\ominus] \\[4pt]
\hat{Y} \xrightarrow{S_L} E^\oplus & & k_3[\hat{Y}][S_L] \\[4pt]
Y \xrightarrow{S_L} E^\ominus & & k_3[Y][S_L] \\[4pt]
Q \xrightarrow{E^\oplus} W^\oplus & & k_4[Q][E^\oplus] \\[4pt]
Q \xrightarrow{E^\ominus} W^\ominus & & k_4[Q][E^\ominus] \\[4pt]
E^\oplus \xrightarrow{E_{decay}} \lambda & & k_5[E^\oplus][E_{decay}] \\[4pt]
E^\ominus \xrightarrow{E_{decay}} \lambda & & k_5[E^\ominus][E_{decay}]
\end{array}
$$

$$(8.36)$$
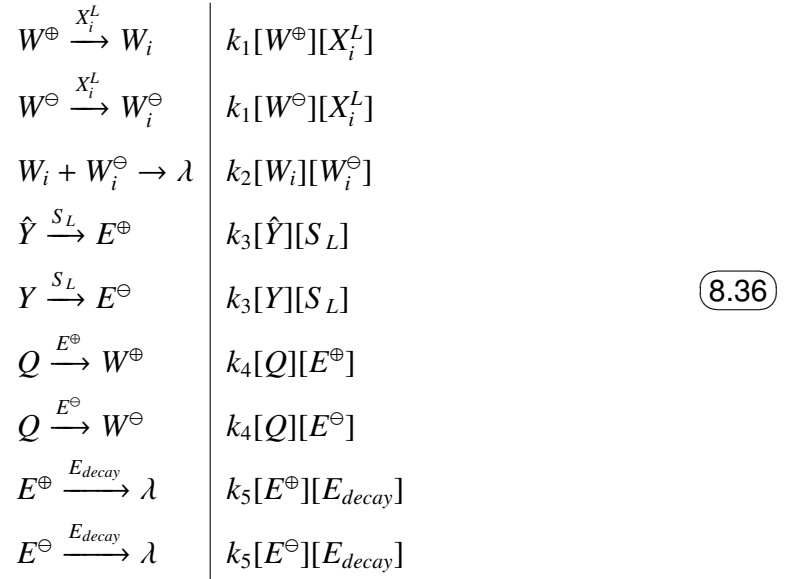
The first three reactions that distribute $W^\oplus$ and $W^\ominus$ are the same as in Section 8.3.1 (illustrated in Figure 8.5). We again omit the reactions 4 and 5 and assume $E^\oplus$ and $E^\ominus$ are given. Let $x_i^L = [X_i^L]$, $y = [Y]$, $\hat{y} = [\hat{Y}]$, $w^\oplus = [W^\oplus]$, $w^\ominus = [W^\ominus]$, $w_i = [W_i]$, and $w_i^\ominus = [W_i^\ominus]$. The corresponding system of ODEs is

$$\frac{dw^{\oplus}}{dt} = k_4 e^{\oplus}$$
$$\frac{dw^{\ominus}}{dt} = k_4 e^{\ominus}$$
$$\frac{de^{\oplus}}{dt} = -k_5 e^{\oplus} e_d$$
$$\frac{de^{\ominus}}{dt} = -k_5 e^{\ominus} e_d.$$

(8.37)

By assuming the starting condition $w_0^{\oplus} = w_0^{\ominus} = 0$ and substituting $e^{\oplus} = -\frac{1}{k_5 e_d}\dot{e}^{\oplus}$ to the first ODE (similarly for $e^{\ominus}$), we obtain

$$w_t^{\oplus} = \frac{k_4}{k_5 e_d}(e_0^{\oplus} - e_t^{\oplus})$$
$$w_t^{\ominus} = \frac{k_4}{k_5 e_d}(e_0^{\ominus} - e_t^{\ominus})$$

(8.38)

Now, because $\hat{Y}$ and Y transform to $E^{\oplus}$ and $E^{\ominus}$ in a one-to-one manner eventually,

$$w_{\perp}^{\oplus} = \frac{k_4}{k_5 e_d}\hat{y}_0$$
$$w_{\perp}^{\ominus} = \frac{k_4}{k_5 e_d}y_0.$$

(8.39)

By setting $k_4 = k_5$ and merging Equation 8.39 with the formulas for the first three (weight distributing) reactions from Equation 8.33, we finalize the weight updating formula of the annealed chemical weight update as

$$\Delta w_i = \frac{x_i^L}{\sum_j x_j^L}(\hat{y} - y)\gamma^{-1},$$

(8.40)

where $\gamma = e_d = [E_{decay}]$ regulates how much of the error $(\hat{y} - y)$ is used for the weight update.

8.4 COMBINING INPUT-WEIGHT INTEGRATION WITH WEIGHT UPDATE

In this section we present different variations of an input-weight integration and weight update. We discuss the analog asymmetric signal perceptron, and introduce the linear chemical perceptron and compare its closed formulas to those of a formal linear perceptron from the theory of neural networks.

In Sections 8.3.1 and 8.3.2 we formalized the direct and annealed weight update in a general form. Recall that a species X_i^L keeps a record of how much the input-weight $X_i - W_i$ contributed to the output Y. As a result of an incorrect output this value is in a linear relation to a weight change. In this section we investigate the impact of the input contribution species on the weight update.

8.4.1 Analog Asymmetric Signal Perceptron

The AASP combines a nonlinear cross-dependent input-weight integration with a direct weight update. The ODE $\frac{dx_i^L}{dt} = k_1 w_i x_i$ for an input contribution species X^L from Section 8.2.1 can be expanded to

$$dx_i^L = -\frac{k_1 w_i}{k_1 w_i + k_2 y} dx_i. \qquad (8.41)$$

Because the right side contains y, which we could approximate only for a single input case, no general formula for x_i^L exists. That is due to the weight cross-dependency. Since X_i^L is produced from X_i, we can bound $x_{i\perp}^L \leq x_{i0}$.

Note that $\frac{dx_i^L}{dt}$ depends on the weight w_i (not just the input x_i), i.e., larger weights produce more x_i^L. Since the direct and annealed weight update Δw_i depends linearly on x_i^L, larger weights get adapted more, i.e., the magnitude of a weight update depends on the weight itself. That means once a weight reaches zero, its concentration cannot be changed

through a learning process anymore, i.e., zero-valued weights are not recoverable.

Since the weight update decreases as the weights approach zero the feedback loop protects the weights from falling too low and reaching zero (a point of no return). However, it holds only if the initial concentration of weights is large enough, $y - \hat{y}$ is within reasonable bounds, and the error decay rate γ is not too small. This potential danger, and often cumbersome setting of parameters, is another reason the AASP (and its cross-dependent input-weight integration) fails as a candidate for a truly robust and general-purpose chemical learning.

Since the AASP is an older model its learning part differs slightly from the general weight update presented here. First, the AASP compares the output Y and the target output $\hat{Y}$ by a rapid annihilation, which redundancy we proven in Section 8.3.1. Second, the AASP uses the learning trigger signal S_L only on the Y-side, which is sufficient. Third, since the AASP employs fixed rate constants optimized by genetic algorithms, it is a specific instance of the nonlinear cross-dependent input-weight integration and the direct weight update. Note that the AASP could use the annealed weight update as well, after a careful setting of γ. Last, its catalytic reactions use Michaelis-Menten kinetics, not the mass-action version.

8.4.2 Linear Perceptron

By a *chemical linear perceptron* (chemical LP) we denote a chemical system with a linear cumulative input-weight integration and a direct or annealed weight update. The input contribution species X_i^L is produced by both positive and negative branches of a linear input-weight integration. Hence, the reactions $X_i \xrightarrow{W_i} Y + X_i^L$ and $X_i \rightarrow Y^{\ominus} + X_i^L$ eventually recycle an input X_i as $x_{i\perp}^L = x_{i0}$. If we denote the initial concentration of an input X_i as x_i, injected at the beginning of the linear cumulative input-weight integration that preceded

the current weight update, the direct and annealed weight update formulas are

$$\Delta w_i = \frac{x_i}{\sum_j x_j}(\hat{y} - y)$$

$$\text{and}$$

$$\Delta w_i = \frac{x_i}{\sum_j x_j}(\hat{y} - y)\gamma^{-1}.$$

(8.42)

This means that, unlike the nonlinear cross-dependent input-weight integration, the input contribution $[X_i^L]$ equals the actual input provided to the linear integration. Since $[X_i^L]$ is detached from a weight concentration, the weights can freely move up or down and even reach zero without imposing a self-feedback.

More importantly, the direct and annealed weight update formulas 8.42 are similar to that of the formal *neural network linear perceptron* (neural network LP)

$$\Delta w_i = x_i(\hat{y} - y)$$

$$\text{and}$$

$$\Delta w_i = x_i(\hat{y} - y)\alpha.$$

(8.43)

The major difference is a normalization of inputs, which for a large number of inputs approaches a constant and could be incorporated into a learning rate. Further, the learning rate α is an inverse of our error decay rate γ, so to have an annealed effect instead of annealing, we (linearly) increase γ over time.

	Chemical Linear Perceptron	NN Linear Perceptron
Input-weight Integration y	$\sum_i (1 - \frac{2}{w_i+1})x_i$	$\sum_i w_i x_i$
Direct Weight Update Δw_i	$\frac{x_i}{\sum_j x_j}(\hat{y} - y)$	$x_i(\hat{y} - y)$
Annealed Weight Update Δw_i	$\frac{x_i}{\sum_j x_j}(\hat{y} - y)\gamma^{-1}$	$x_i(\hat{y} - y)\alpha$

Table 8.1: Comparison of the input-weight integration and weight update formulas of the linear chemical and formal neural network perceptrons.

8.5 RESULTS

In this section we compare the performance of the chemical linear perceptron, its neural network counterpart, and the AASP. We proved mathematically that the differential equations obtained from the reactions responsible for the linear input-weight integration and weight update collapse to simple closed formulas. Our simulations confirm these theoretical results as illustrated in Figure 8.7. Because the behavior of the analytic and the ODE version of a chemical linear perceptron match perfectly, we evaluate the performance of the analytic version only. Note that the simulation cost of the analytic version is fractional—roughly 1000 times faster than the ODE numerical integration using a Runge-Kutta4 method with 0.05 time step over 800 training iterations.

To provide a broad view on the learning capabilities of the aforementioned models, we use 6 linear and nonlinear target functions (of two inputs) from Chapter 5 and 4 time series from Chapter 6. We evaluate the performance calculated as the RNMSE and SAMP error over 10,000 runs, each consisting of 800 training iterations. For all the tasks we draw initial weight concentrations uniformly from the interval $(0.5, 2)$ for the chemical LP, $(-0.5, 0.5)$ for the neural network LP, and $(0.5, 1.5)$ for the AASP.

It is important to recall that the input-weight integration of the chemical LP (Equations 8.24 and 8.25) equals that of the neural network LP (Equation 8.26) with the weights restricted to the range $(-1, 1)$. In other words, any weight setting of the chemical LP could be translated to an equivalent setting for the neural network LP. Thus, technically, the representation power of the neural network LP is greater than that of the chemical LP. In an idealized learning process the final performance of the neural network LP would not be smaller, however, a practical realization of learning and the weight update tuned by, e.g., an initial weight distribution and learning annealing rate, greatly affects the weight
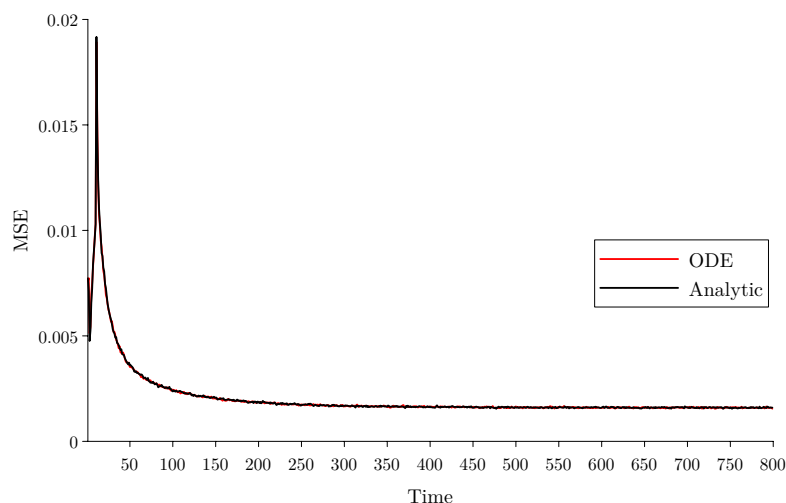
Figure 8.7: Mean square error (MSE) of the analytic and the ODE version of a chemical linear perceptron on the NARMA10 task using a delay line of size 10 averaged over 10,000 runs, each consisting of 800 learning iterations. The analytic and ODE versions match perfectly.

convergence, and in our simulations often results in performance that is (slightly) worse than what we obtained in chemistry. Note that we selected the target functions and time series such that the minimal and maximal expected output is within bounds enforced by the $(-1, 1)$ weight restriction. Generally, fitting the weights to the $(-1, 1)$ range requires statistical knowledge of the target task and appropriate scaling of the inputs. On the other hand, since the effective weights are bounded their convergence is "safer" than of the neural network perceptron. Unlike the chemical model, the weights of the neural network counterpart are unbounded, hence they can diverge to infinity and the setting of the initial learning and annealing rate is more sensitive to extremal events.

The chemical LP's error decay rate γ, an inverse of the formal learning rate, is the concentration of the species E_{decay}. For each task we combined initial γ of 0.1 and 0.05 with four "an annealing rates" of $0.0001, 0.0005, 0.001$, and 0.005, which are used to increase γ each learning iteration. In this section we report the best results only. Note that a manual increment of γ in a wet chemical experiment could be replaced by a constant

influx of E_{decay} at a prescribed rate. The neural network LP uses a linear annealing of the learning rate α such that it reaches zero at 800th learning iteration. Again we optimized the learning rate for each task individually using the values $0.1, 0.2, \ldots, 0.7$.

8.5.1 Static Functions of Two Inputs

The chemical LP reaches a marginal error on all static linear functions of two inputs listed in Table 5.2 : RNMSE of 0.0007 to 0.0158 and SAMP of 0.003 to 0.13 (Figure 8.9 and Table 8.2). As expected, performance significantly decreases for the quadratic (nonlinear) function $kx_1x_2 + k_0$ (RNMSE of 0.235, SAMP of 2.38). The learning error of the chemical

(a) RNMSE

(b) SAMP

Figure 8.8: Final RNMSE and SAMP of the chemical linear perceptron (CHLP), the neural network linear perceptron (NNLP), and the AASP after 800 learning iterations for 6 linear and nonlinear functions of two inputs averaged over 10,000 runs.

linear perceptron is an order of magnitude smaller compared to the AASP (Figure 8.8); for the functions $k_1 x_1$ and $k_2 x_2$, the ratio is 355 for RNMSE and 1834 for SAMP. Note that the AASP cannot fully eliminate the contributions (or consumptions) of the superfluous inputs, such as the input X_2 for the target function $k_1 x_1$. Even though the AASP's input-output relation is nonlinear and would seem to be better suited for a quadratic function $k x_1 x_2 + k_0$, the chemical linear perceptron reaches 30% smaller RNMSE and 50% smaller SAMP on that task. The performance of the chemical and neural network linear perceptrons on a quadratic function match almost perfectly (1–6% difference) and approach the best achievable error represented by linear regression (Table 8.2). On the linear functions the neural network LP outperforms the chemical LP on average if the error is measured by RNMSE, however, the situation is opposite for the SAMP. In general the representation potential and learning capabilities of these two models are coherent and the small

Table 8.2: Final RNMSE and SAMP of the chemical linear perceptron (CHLP), the neural network linear perceptron (NNLP), the linear regression (L Reg), and the AASP after 800 learning iterations for 6 linear and nonlinear functions of two inputs averaged over 10,000 runs. The values are rounded to 5 decimal places.

(a) RNMSE

Name	AASP	CHLP	NNLP	L Reg
$k_1 x_1 + k_2 x_2 + k_0$	0.10344	0.00157	0.00036	1.98×10^{-15}
$k_1 x_1 - k_2 x_2 + k_0$	0.31045	0.01236	0.00082	1.82×10^{-15}
$k x_1 x_2 + k_0$	0.30424	0.23486	0.23313	0.22664
$k_1 x_1$ and $k_2 x_2$	0.25162	0.00071	0.00054	1.91×10^{-15}
k_0	0.37839	0.01582	0.00160	8.62×10^{-15}

(b) SAMP

Name	AASP	CHLP	NNLP	L Reg
$k_1 x_1 + k_2 x_2 + k_0$	1.23594	0.00546	0.00372	2.23×10^{-14}
$k_1 x_1 - k_2 x_2 + k_0$	3.78678	0.12593	0.00992	2.19×10^{-14}
$k x_1 x_2 + k_0$	3.68816	2.38396	2.52996	2.47268
$k_1 x_1$ and $k_2 x_2$	5.98028	0.00326	0.01339	2.86×10^{-14}
k_0	6.75389	0.07327	0.01909	1.23×10^{-13}

performance disparities are an artifact of different annealing methods and parameters.

(a) RNMSE

(b) SAMP
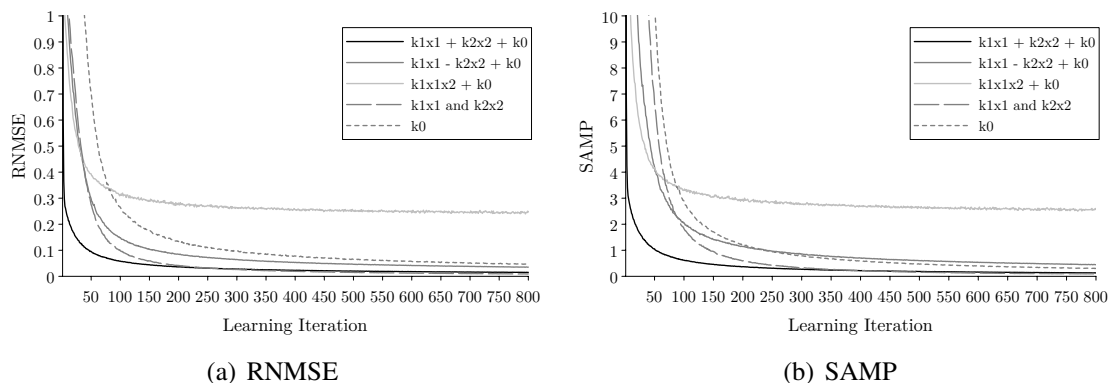
Figure 8.9: RNMSE and SAMP of the chemical linear perceptron for 6 linear and nonlinear functions of two inputs averaged over $10,000$ runs with the initial $\delta = 0.1$ and 0.001 increment (annealing).

8.5.2 Time Series

To investigate the scalability of the AASP as well as the chemical and neural network linear perceptrons, we use these models for 2 to 20 inputs and integrate them with a delay line of appropriate length. For the AASP, we opted for an algorithmic rather than chemical delay line because of the simulation cost and precision. Note that the number of species and reactions of the best performing and most reliable chemical delay line, called the parallel-accessible delay line (PDL), grows quadratically. Also, since the AASP's input species annihilate with the output and the PDL's production of the past inputs is non-instant, to a small extent, the AASP's performance could be negatively affected. Since our goal here is not to test the integration of a chemical delay line with chemical learners but rather to investigate the learning capabilities of different models, we opted for a non-chemical (algorithmic) delay line, which emulates an instant feeding of the past inputs. Note that since the input-weight integration of the chemical linear perceptron is cumulative, using a chemical delay line with any latency (such as the manual signalling variant)

would yield the same results (not presented here).

The final errors for all the target time series are summarized in Figure 8.10 and Table 8.3. The linear chemical perceptron compared to the AASP reaches significantly higher performance on the four time series, the linear-weighted moving average of order two (LWMA2), the weighted moving maximum of order two (WMM2), NARMA2 and NARMA10. For the LWMA2, the final error of the chemical linear perceptron out of all delay line lengths is very low and reaches an RNMSE of 0.004 and a SAMP of 0.02, which is 29, respectively 57 times lower compared to the AASP. Performance on the most difficult, highly nonlinear NARMA10 task is naturally worse (the RNMSE of 0.346, the
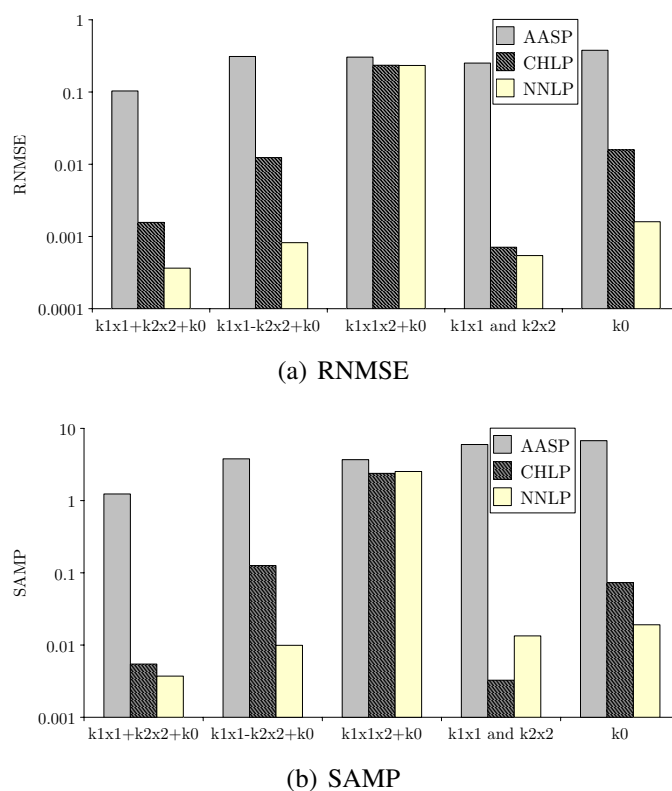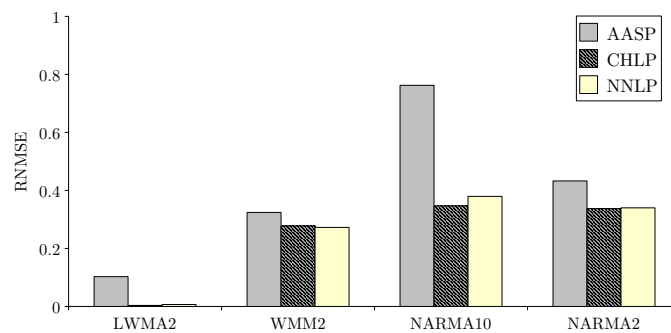

(a) RNMSE
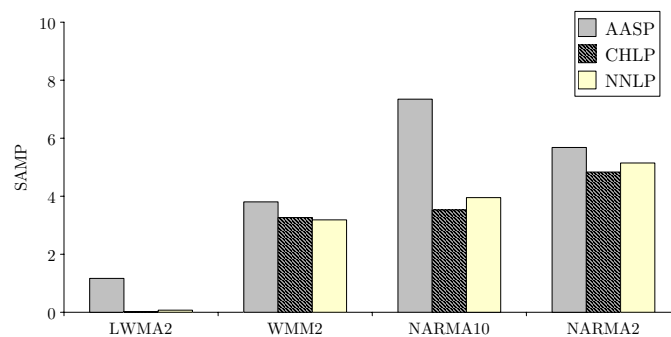
(b) SAMP

Figure 8.10: Final RNMSE and SAMP of the chemical linear perceptron (CHLP), the neural network linear perceptron (NNLP), and the AASP after 800 learning iterations for the 4 target times series averaged over 10,000 runs. The best global results obtained using the delay line of size 2 to 20 are shown.

SAMP of 3.53), however that reduces the AASP's error roughly by a factor of two. As for the static linear and nonlinear functions, the neural network linear perceptron performs similarly to the the chemical one. For the nonlinear time series, i.e., WMM2, NARMA2, and NARMA10, the difference is within 12%.

Since the LWMA2 and WMM2 tasks are determined purely by the last two inputs, the expected optimal delay line (memory) size is two. Because the NARMA2 and NARMA10 tasks are recurrent, their calculation is based on the last 4 and 20 inputs respectively. The performance of the linear chemical and neural network perceptrons are in line with these values. The results also show that any extra past input that required by the task, which basically acts as noise, is effectively discarded with a small impact on performance. On the other hand, a nonlinear cross-dependent input-weight integration employed by the AASP interlinks the contributions of the weights as we showed theoretically. This is not a major issue for small system sizes, but the larger the number of inputs, the less distinct the weights. As a consequence, the performance drops sharply. As shown in Figure 8.11, the scalability of the AASP is poor.

8.6 DISCUSSION

By applying an analytic approach we derived the closed formulas for both the input-weight integration and weight update of chemically simulated learning. We introduced a new, cumulative input-weight integration, which we employed in the design of the chemical linear perceptron. To mediate the weight convergence, we implemented an annealed weight update controlled by the concentration of the error decay species. It amplifies or reduces the transformation of the output and the target output species to the weight-changers, which are split among participating weights proportionally. The analytic solu-
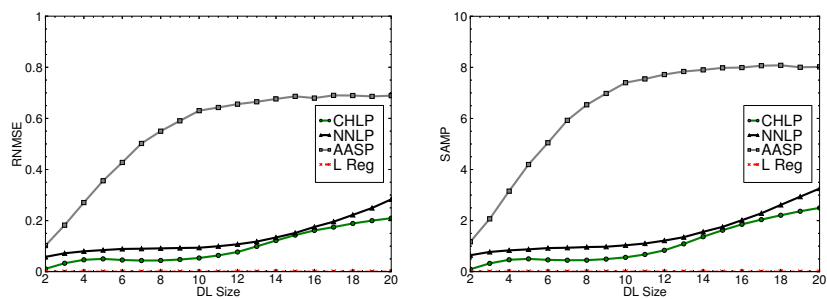
(a) LWMA2
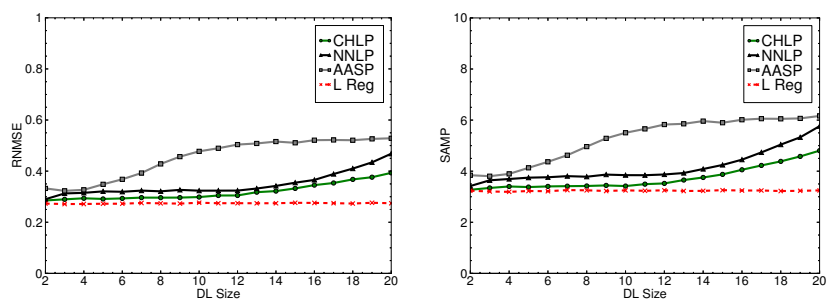
(b) WMM2
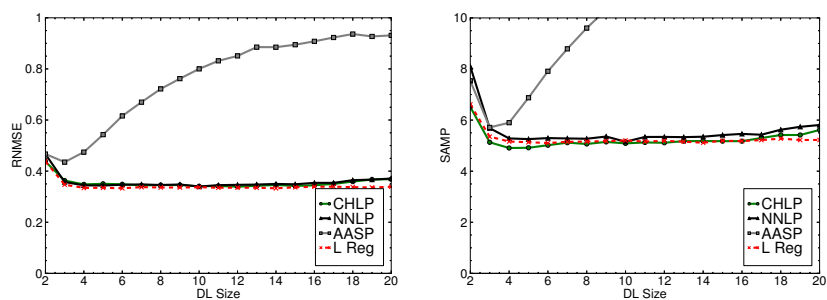
(c) NARMA2
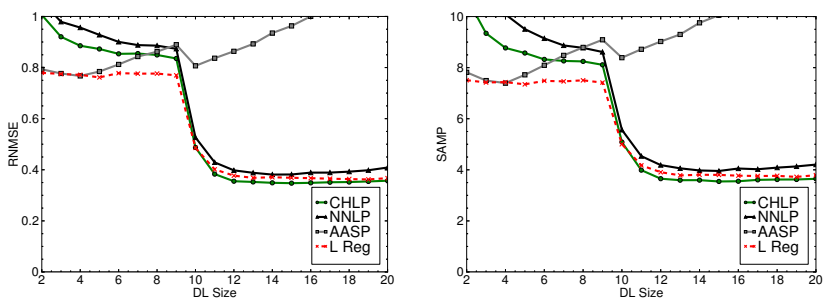
(d) NARMA10

Figure 8.11: Final RNMSE and SAMP of the chemical linear perceptron (CHLP), the neural network linear perceptron (NNLP), the linear regression (L Reg), and the AASP after 800 learning iterations for the 4 target times series averaged over 10,000 runs and the delay line sizes from 2 to 20. The most scalable results are shown for the CHLP and the NNLP out of all learning/annealing rate combinations.

Table 8.3: Final RNMSE and SAMP of the chemical linear perceptron (CHLP), the neural network linear perceptron (NNLP), the linear regression (L Reg), and the AASP after 800 learning iterations for 4 target times series averaged over 10,000 runs. The values are rounded to 5 decimal places.

(a) RNMSE

Name	AASP	CHLP	NNLP	L Reg
LWMA2	0.10274	0.00356	0.00672	1.80×10^{-15}
WMM2	0.32412	0.27743	0.27250	0.27015
NARMA10	0.76234	0.34641	0.37947	0.36334
NARMA2	0.43269	0.33732	0.33992	0.33193

(b) SAMP

Name	AASP	CHLP	NNLP	L Reg
LWMA2	1.16794	0.02066	0.07277	2.07×10^{-14}
WMM2	3.80595	3.26660	3.18554	3.19002
NARMA10	7.34650	3.52902	3.95260	3.73775
NARMA2	5.68125	4.83426	5.14635	5.10683

tions provided insight into the functioning of the linear chemical perceptron and placed it solidly side by side to its neural network counter part. We verified the equivalence between the analytic and ODE-based version and thus justified the use of the analytic version. A practical significance of this work translates into a huge saving in simulation time. This enables to transition to more advanced chemical constructs, such as feedforward multi-compartment chemical neural networks, which have been so far too time consuming to investigate in depth. This work bridged adaptive CRNs with neural networks and opened a new territory for further exploration of chemical learning. Our analytic approach paid well also in significantly lower learning error (up to 437 times), compared to the AASP. We showed that the AASP does not scale due to a cross-dependent input-weight integration.

The size of the chemical linear perceptron with 2 inputs (and bias) is relatively small: 22 reactions and 21 species. To simplify the construction, the production of species X_i^L by the input-weight integration reactions could be removed and provided alongside the

target output at the same concentration as preceding input.

The linear chemical perceptron can work with arbitrary rate constants that obey the inherent system symmetries. The design is therefore robust. In fact, for the very first time we could avoid choosing rate constants empirically or using genetic algorithms. We only fixed the rate constants of the input-weight integration reactions more for convenience than necessity. By setting the rates of the reactions that transform the input to Y and $\hat{Y}$ to the value one, we defined the weight positive region for the concentration > 1 and negative for < 1.

Compared to the neural network version, the performance of the linear chemical perceptron is equivalent on average, however, for the nonlinear functions (time series) is slightly better (even if compared to linear regression). That might seem surprising since the weights of the neural network perceptron are unbounded and could represent a wider range of functions. In fact, since the effective weights of the chemical linear perceptron are restricted to the $(-1, 1)$ interval, a weight search becomes easier, assuming the (sub)optimal weights lie in this region, which we can deduct from the target task formula. The learning process implementing an input-normalized delta rule is more effective and safer, since a divergence is limited. On the other hand, a selection of the target functions must take this limitation into consideration, and if needed, the inputs must be rescaled. To conclude, the formal neural network linear perceptron is more universal, but for the price of a potentially unbounded search.

We demonstrated the learning capabilities of the linear chemical perceptron on 6 static linear and nonlinear functions of two inputs as well as 4 time series. On the all linear functions the RNMSE error reached low values below 0.0158. For the most difficult time series, NARMA2 and NARMA10 tasks, the RNMSE rose to 0.337 and 0.346 respectively. Note that the NARMA task is generally tackled with a magnitude larger and more

complicated machine learning approaches than our single memory-enabled chemical perceptron. A common approach is to employ recurrent neural networks [15] or advanced reservoir computing [33], i.e., echo state networks [78] or liquid state machines [99], consisting of hundreds of nodes (neurons). For instance, minimal complexity echo state networks reported by Tino and Rodan [129] needed 50 nodes to achieve a NMSE of 0.16, i.e., RNMSE of 0.4 (higher than for a linear perceptron with a delay line).

9

BIOCHEMICAL IMPLEMENTATION

As specified in Section 2.3.1 CRNs describe the behavior of chemical system in terms of reactions and kinetic rates without any assumption about the molecular structure of the chemical species. In CRNs the reactions specify how the molecules should interact to achieve the desired behavior, such as learning or memorization, but they do not tell which molecular structures out of the pool of possible wet chemicals could carry out these dynamics. The symbols, e.g., X, Y, and W_0, are, therefore, placeholders for any species that would obey the defined interaction constraints and kinetics. We considered the basic characteristics of "well-behaved" reactions, such as that the maximal number of reactants is two. Also, we minimized the number of species and reactions of each CRN to simplify eventual wet biochemical implementations.

A system-level abstraction of CRN allowed us to explore and better understand the species roles and interaction principles of a wide range of models. Keeping CRN species symbolic preserved generality. In theory, a single CRN could serve (and be mapped to) several sets of wet chemical substrates, i.e., symbolic species' substitutions. CRN is often perceived as a high-level chemical programming language. For computer scientists, it could be viewed as a *pseudocode* holding the essential properties of an algorithm without being corrupted by the constructs provided by a specific language, such as Java or C++.

The mapping from CRN to naturally-occurring or synthetic chemicals is nontrivial.

For a CRN consisting of a single reaction $2X + Y \rightarrow 2Z$ we can substitute the species X to diatomic hydrogen H_2, the species Y to diatomic oxygen O_2, and the species Z to hydrogen oxide H_2O (water), and thus obtain a wet chemical implementation of the symbolic reaction (without the mapping of rate constants). Even though such ad-hoc mappings might exist for simple CRNs, they are not extendible and they often require intimate knowledge about the target substrate molecules. In order to accommodate a potentially unlimited number of species, ideal substrate molecules should be "concatenable" without bounds to create longer and longer polymers from a given set of monomers. Also, it is essential that the outcomes of reactions are predictable just from the structure of the reactants. Although several chemistries satisfy these conditions, the most popular used molecule in biochemistry capable of polymerization with highly predictable results is DNA (deoxyribonucleic acid). We will therefore limit our biochemical endeavours to DNA chemistry.

DNA, whose 3D structure was described by Watson and Crick in 1953 [158], is a molecule that carries genetic information of all living forms, and is thus referred to as a "molecule of life". As shown in Figure 9.1, DNA consists of 4 nucleotides (bases): adenine (A), cytosine (C), guanine (G), and thymine (T). DNA has a double helix structure, which is due to the bindings of adenine with thymine (A-T) and guanine with cytosine (G-C) through hydrogen bonds. This binding mechanism is called Watson-Crick base pairing or Watson-Crick complementarity. Given a nucleotide sequence, there is only one inverse sequence fully matching it. Because DNA chemistry is structured and we can concatenate DNA sequences into longer, linear or nonlinear structures, we could create potentially an unlimited number of different molecular species. That gives us a huge modeling power and control over reaction design.

Despite the beneficial properties of DNA, moving directly to raw DNA sequences
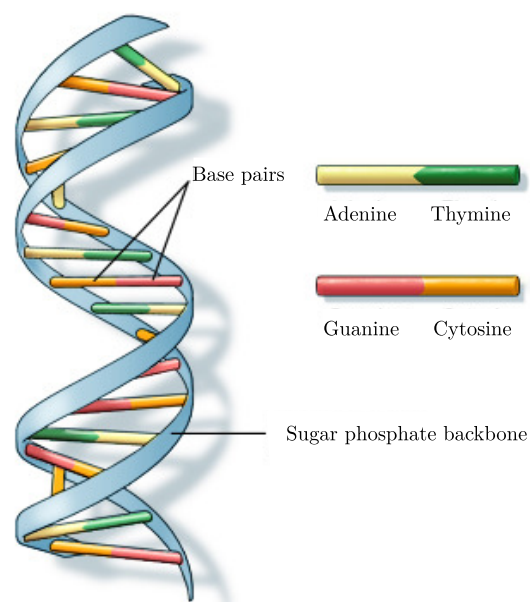
Figure 9.1: DNA structure (adapted from US National Library of Medicine).

would not be feasible. To make the mapping process gradual, we apply standard intermediate abstraction levels, as shown in Figure 9.2, commonly used in DNA chemistry. This general roadmap [138, 165] starts with a formal model described by a set of equations or diagrams for the functioning of the system of interest. The next step is a CRN specification with species and reactions producing the required output for given inputs as prescribed by the model. Moving from a formal model to a CRN is not standardized and its difficulty varies greatly. After we obtain a CRN, an ideal and the most general chemical model, we create a new equivalent CRN compatible with DNA primitives that we want to map our species to. That includes introducing new species, reactions, and replicating and splitting the existing reactions into several intermediates. The original CRN thus gives a lower bound on the system size. After a CRN is in right format, we map species to DNA domains. At this level of abstraction we specify the spatial organization of DNA species

1. Formal Model $\sum_i w_i x_i > \theta$

2. CRN $X_1 \rightarrow X_3 + X_2$

3. Translated CRN $X_1 + G_i \rightarrow W_{1,i} + O_i$

$O_i + T_i \rightarrow W_{2,i} + X_2 + X_3$

4. DNA Domains

5. DNA Sequences

CGTCGCGACGGTCGCA

GCAGCGCTGCCAGCGT
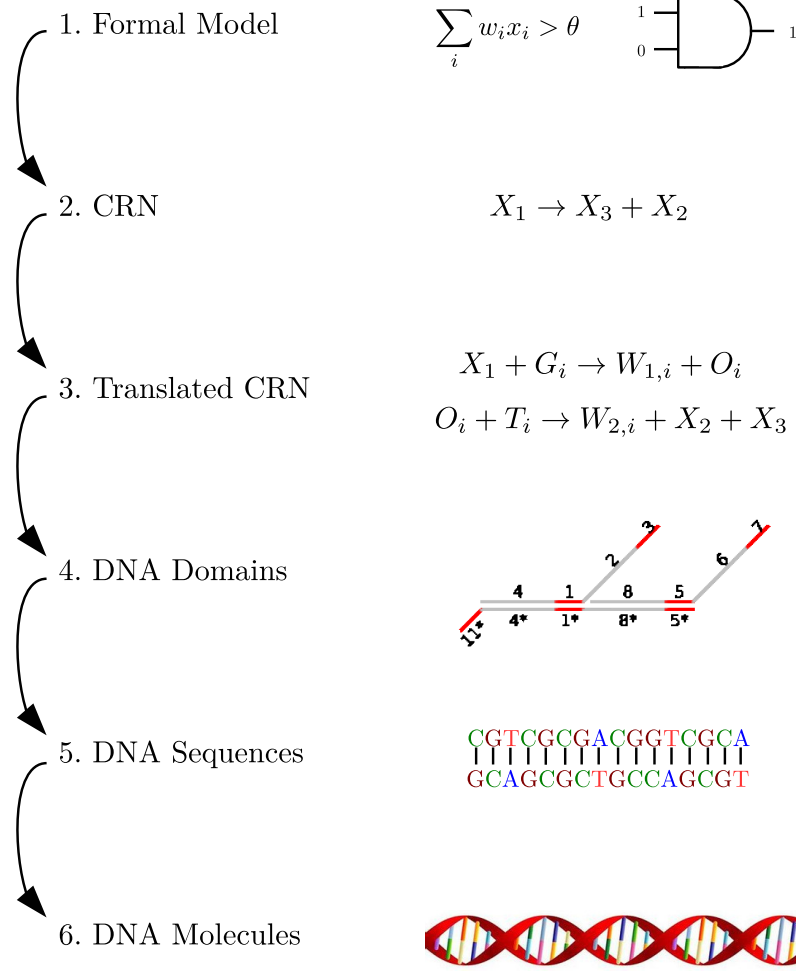
6. DNA Molecules

Figure 9.2: A roadmap showing different abstraction levels going from a formal model through CRN specification and to DNA domains, DNA sequences, and ultimately (wet) DNA molecules.

and logically decompose them into shared parts, so-called domains. A domain is a subsequence of single-stranded DNA labeled with a number. Domains are classified as long, typically consisting of more than 20 bases, and short with 5 bases (called toehold or sticky end). At the domain level we abstract from actual sequences and assume that domains are unique and that they bind only to their complements distinguished by an asterisk *. For instance, a single strand decomposed into domains 1 and 2 binds with a complementary

single strand 1^*-2^*. Further, because of the length, a binding of long strands is more stable and stronger than of short strands.

In the following sections we present two options for transforming a CRN to a chemistry specified with DNA domains: DNA-strand displacement and deoxyribozymes. For the first option there exists a generic compilation process, which can transform any CRN to an adapted CRN (step 3) and associate DNA domains to the species with a minimal effort. Even though a transformation to deoxyribozymes is not automatic and requires ad-hoc adjustments of the original CRNs, compared to DNA-strand displacement, this method is better suited for catalytic reactions and requires fewer species and reactions. To demonstrate our CRN models are wet-implementable, we illustrate a DNA-strand displacement and deoxyribozyme implementation for the linear chemical perceptron and the manual signalling delay line.

The next step, turning domain-specified DNA molecules into nucleotide sequences (e.g., A-T-C-C-T) is provided by sequence designer tools, which apply combinatorial designs to produce the sequences of required length, satisfying the complementarity of domains with a minimal overlap, and binding among non-complementary domains. The most popular DNA sequencer with a convenient web interface, which (partially) addresses undesired binding (crosstalk or leakage), is NUPACK [164].

The last step, i.e., a synthesis of the sequences and producing wet DNA molecules is supplied by several companies. Ideally, if all the mappings between the adjacent abstraction levels are performed accurately, an experiment conducted with the synthesized DNA molecules should yield the same result as a simulated CRN. We will discuss the source of potential discrepancies later.

9.1 DNA STRAND DISPLACEMENT

DNA strand displacement [163, 166, 167] is a simple, yet powerful reaction primitive, which operates on single and double DNA strands. Its most advanced applications include a square root calculation [127] and a construction of Hopfield network [127].

Single strands are classified as upper or lower, indicating their position if bound in a double strand. Although DNA strands could be flipped, so technically each upper strand could be a lower strand and vice versa, for convenience, we assume that the strands composed of the domains labeled with plain numbers (1, 2 etc.) occur in an upper position and the strands consisting of complementary domains ($1^*, 2^*$ etc.) occur in a lower position, and this distinction is exclusive (no mixed domains).

The orientation of a DNA molecule is determined by its 5′-end and 3′-end. Two complementary strands bind if their directions (and ends) are opposite. Using the upper/lower distinction upper strands are right handed (e.g., 5′-1-2-3′) and lower strands are left handed (e.g., 3′-1^*-2^*-5′). In all our figures long domains are gray, short domains are red and strands' 3′ and 5′ ends are implied from their upper/lower position.

What we refer to as a full double strand is a perfect Watson-Crick DNA double strand, where all bases pairs of corresponding upper and lower strands are complementary. Partial double strand is similar to a full double strand but instead of all pairs, only a substring of upper and lower strands matches. Full or partial double strands, or more complicated nonlinear DNA structures, such as deoxyribozymes (Section 9.2), are called complexes or gates.

In a basic version of DNA strand displacement, with a full name toehold-mediated DNA strand displacement, two species, a single strand X and a complex G, react. As shown in Figure 9.3, the toehold domain 1^* of a double strand G is unmatched and binds

(a) $X + G \leftrightarrow XG$

(b) $XG \leftrightarrow GX$

(c) $GX \leftrightarrow H + Y$
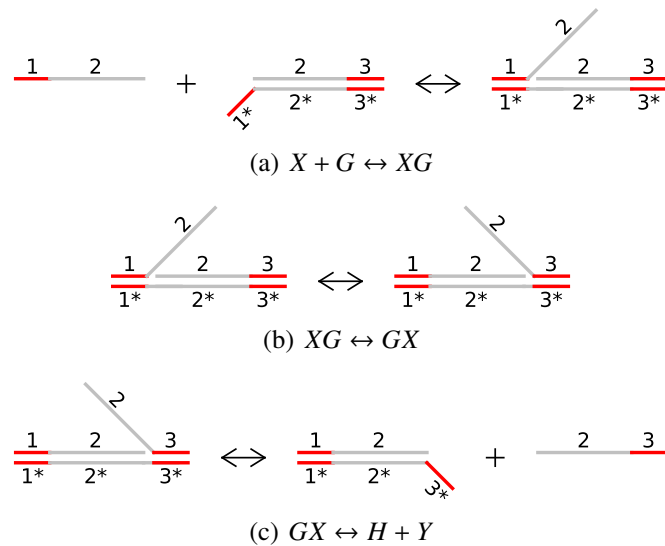
Figure 9.3: DNA strand displacement—an upper strand *X* displaces a strand *Y* from a complex *G* in a series of reactions. Long domains are gray and short domains are red.

to its complementary domain 1 of a single strand *X* creating the complex *XG*. Then, the domain 2 of an intruding upper strand *X* and the identical domain 2 of an original complex *G* compete over the complementary domain 2* since they are both in a binding position. In the process called branch migration, a crossing point of the two competing domains moves stochastically left and right between the domains 1 and 3 as a random walk. Once the crossing point reaches the rightmost position, the domain 2 of the strand 2-3 is partially displaced by the intruding domain 2. Finally, the toeholds 3 and 3* separate and the strand 2-3 labeled as *Y* is fully displaced and detached from the complex. Note that these reactions are bidirectional, i.e., a single strand *Y* could displace *X* from a complex *H*. The reaction rates depend on the domain lengths and toehold binding strength. Also, if the domains 3 and 3* were removed, the final complex *H* would become a full double strand and the reaction $GX \rightarrow H + Y$ would be effectively unidirectional.

9.1.1 CRN to DNA-Strand Displacement Compilation

In this section we introduce Soloveichik's method [140], which compiles a CRN to a DNA strand displacement circuit "automatically." Soloveichik proved that a strand displacement circuit can approximate, with arbitrarily small error, any CRN-based solely on mass action kinetics. Since this method transforms species, reactions, as well as rates universally it could be applied to each of our models, once the catalytic reactions $S \xrightarrow{E} P$ using Michaelis-Menten kinetics are flattened to mass action $S + E \rightarrow P + E$. Besides the mass action requirement, each reaction must have either one or two reactants, which we comply with by default. Note that there is no restriction on the number of products.

The compilation uses several extended variations of the basic DNA strand displacement (Figure 9.3). Several DNA strand displacement reactions cascade in a chain, where a product of one is a reactant of the next displacement. Populations of the original CRN species, the signals, are represented by the populations of single-stranded molecules, each consisting of four unique domains (two short and two long). Since the signal DNA strands are entirely distinct, they do not interaction with each other, but their transformation is mediated by double-stranded complexes.

A unimolecular reaction with a reactant (substrate) S, products $P_1, \ldots, P_n$, and rate constant k is transformed to two cascaded displacements as

$$
\mathbf{S} \xrightarrow{k} c_1 \mathbf{P_1} + \ldots + c_n \mathbf{P_n} \quad \rightsquigarrow \quad
\begin{cases}
\mathbf{S} + G \xrightarrow{q} W_1 + O \\[2mm]
O + T \xrightarrow{q_{\max}} W_2 + c_1 \mathbf{P_1} + \ldots + c_n \mathbf{P_n}
\end{cases}
, \quad \boxed{9.1}
$$
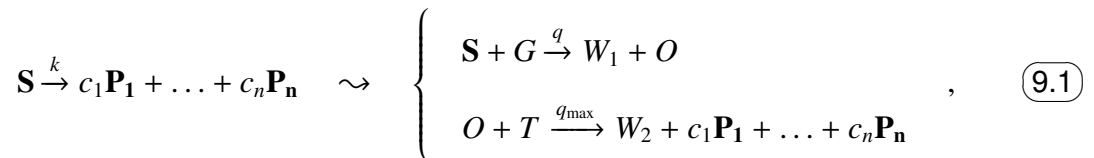
where G, T, O, W_1, W_2 are new intermediate species and q and q_{max} are new rate constants (which will discuss later). Initially a single strand S displaces O from a complex

(a) $S + G \rightarrow W_1 + O$

(b) $O + T \rightarrow W_2 + \mathbf{P_1}$
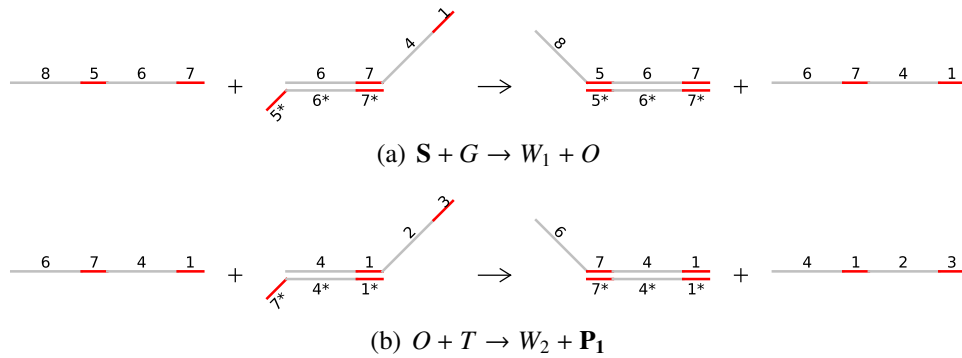
Figure 9.4: Transformation of a formal unimolecular reaction with one product $\mathbf{S} \rightarrow \mathbf{P_1}$ into a cascade of two DNA strand displacements. The original species (in bold) are single stranded molecules consisting of four unique domains. Long domains are gray and short domains are red.

(a) $S + G \rightarrow W_1 + O$

(b) $O + T \rightarrow W_2 + \mathbf{P_1} + \mathbf{P_2}$
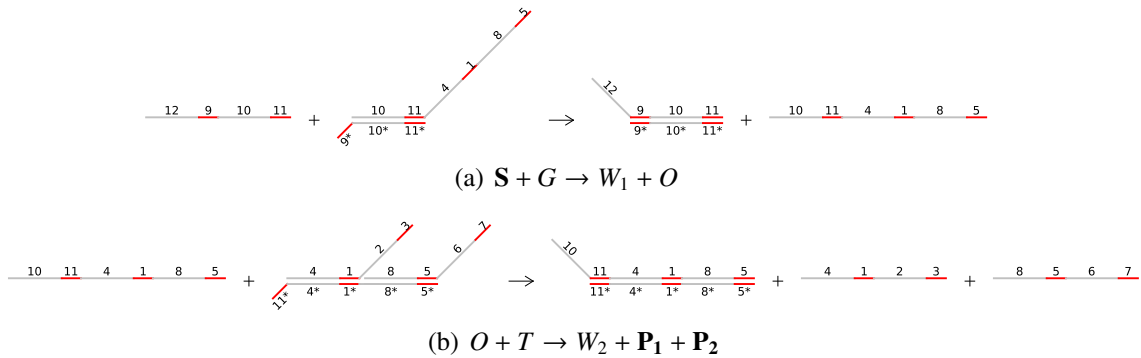
Figure 9.5: Transformation of a formal unimolecular reaction with two products $\mathbf{S} \rightarrow \mathbf{P_1} + \mathbf{P_2}$ into a cascade of two DNA strand displacements. The original species (in bold) are single stranded molecules consisting of four unique domains. Long domains are gray and short domains are red.

G producing waste W_1. Once a single strand O is produced, it displaces the products $\{P_i\}$ from a complex T producing another double-stranded waste W_2. Note that G and T are fuel species, which are supplied at large concentrations, higher than the substrate concentration $[S]$. This ensures that the reaction rates effectively become constant over the required lifetime.

The transformation of an unimolecular reaction with a single product is illustrated in Figure 9.4, and a two-product version in Figure 9.5. For a decay reaction with no product,

$S \rightarrow \lambda$, the second superfluous displacement is removed, i.e., both W_1 and O are wastes.

A bimolecular reaction with reactants (substrates) S_1 and S_2, products $P_1, \ldots, P_n$, and rate constant k is transformed to three cascaded displacements as

$$\mathbf{S_1} + \mathbf{S_2} \xrightarrow{k} c_1 \mathbf{P_1} + \ldots + c_n \mathbf{P_n} \quad \rightsquigarrow \quad \begin{cases} \mathbf{S_1} + L \xleftrightarrow[q_{max}]{q} H + B \\[2ex] \mathbf{S_2} + H \xrightarrow{q_{max}} W_1 + O \\[2ex] O + T \xrightarrow{q_{max}} W_2 + c_1 \mathbf{P_1} + \ldots + c_n \mathbf{P_n} \end{cases} \quad , \quad \boxed{9.2}$$

where similarly to the unimolecular case, L, H, B, O, T, W_1, W_2 are new intermediate species, and q and q_{max} are new rate constants.

Initially, a single strand S_1 displaces B from a complex L producing a double strand H. Bear in mind that this reaction is bidirectional, since a lower toehold of double strand H is naked (the toehold 9^* and 13^* shown in Figure 9.6 and 9.7 respectively). Double strand

(a) $\mathbf{S_1} + L \leftrightarrow H + B$

(b) $\mathbf{S_2} + H \rightarrow W_1 + O$

(c) $O + T \rightarrow W_2 + \mathbf{P_1}$
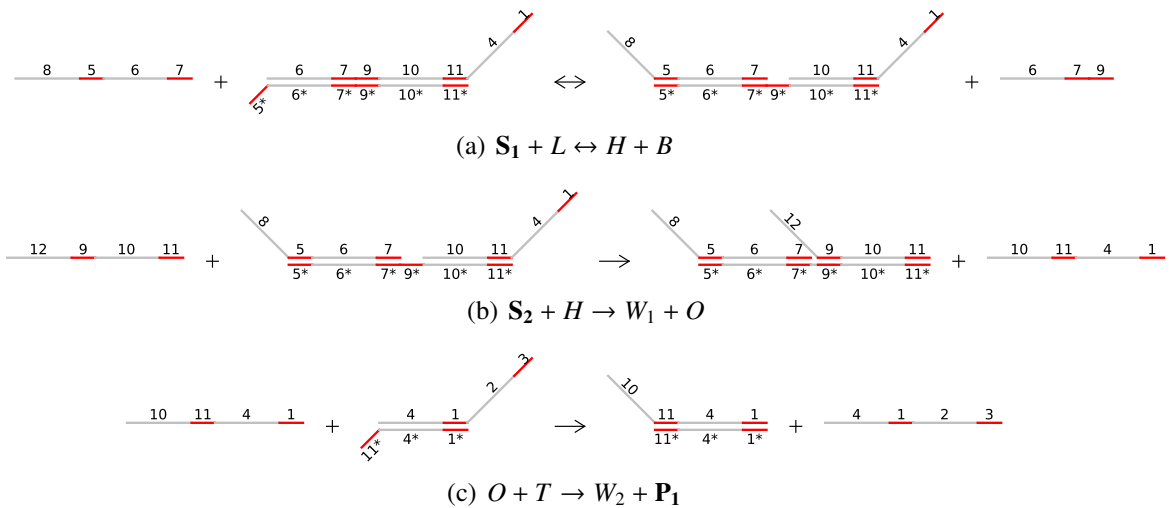
Figure 9.6: Transformation of a formal bimolecular reaction with one product $\mathbf{S_1} + \mathbf{S_2} \rightarrow \mathbf{P_1}$ into a cascade of three DNA strand displacements. The original species (in bold) are single stranded molecules consisting of four unique domains. Long domains are gray and short domains are red.

(a) $\mathbf{S_1} + L \leftrightarrow H + B$

(b) $\mathbf{S_2} + H \rightarrow W_1 + O$

(c) $O + T \rightarrow W_2 + \mathbf{P_1} + \mathbf{P_2}$
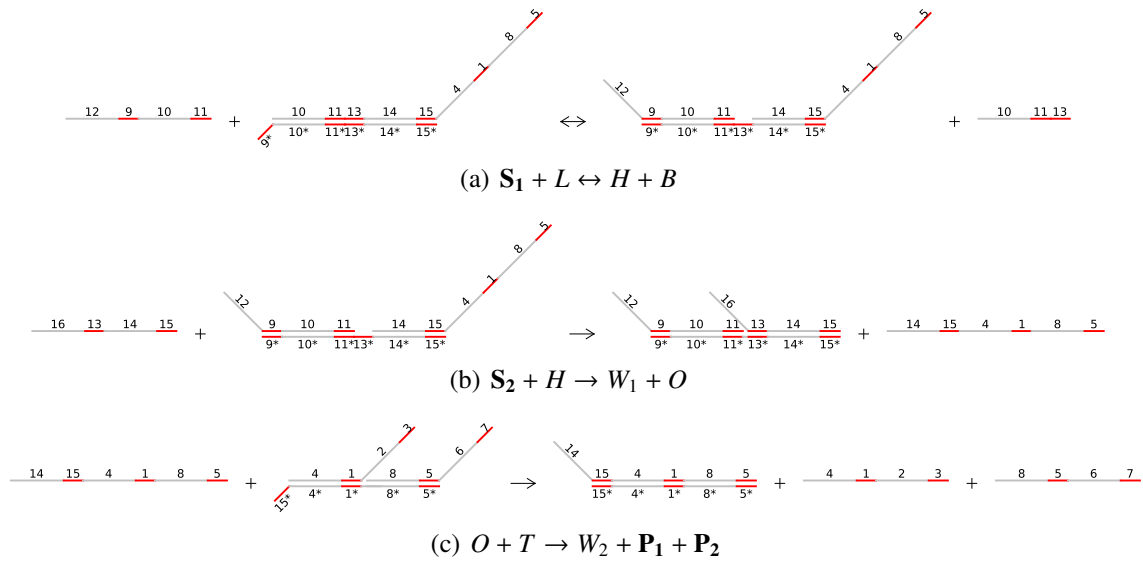
Figure 9.7: Transformation of a formal bimolecular reaction with one product $\mathbf{S_1} + \mathbf{S_2} \rightarrow \mathbf{P_1} + \mathbf{P_2}$ into a cascade of three DNA strand displacements. The original species (in bold) are single stranded molecules consisting of four unique domains. Long domains are gray and short domains are red.
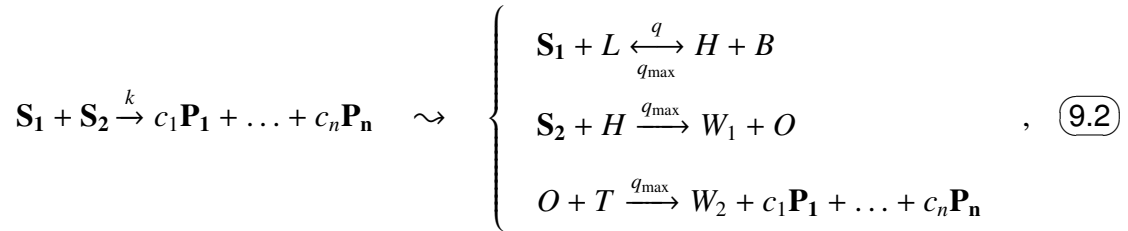
H then cascades to the second reaction where its upper strand O detaches by the second signal, a single strand S_2. Strand O and complex T then produce the required products. As in the unimolecular case, the species W_1 and W_2 are inert waste species, which serve no purpose, other than being produced. The species L, B, and T are fuel species supplied at large concentrations. Also, for an annihilatory reaction with no product $S_1 + S_2 \rightarrow \lambda$, the third superfluous displacement is removed, i.e., both W_1 and O are waste.

As seen from the construction, Soloveichik's transformation produces a vast amount of intermediate fuel, mediating, or waste species. That is due to a single-stranded structure of the formal species, called signals. In some situations, specific signals can be mapped to double strands, thus saving intermediates. An example is a system consisting of a single reaction $S_1 + S_2 \rightarrow P_1 + P_2$, where S_1 and P_2 could be mapped to single strands and S_2 and P_1 to double strands. In the general case, where a single species is a reactant or a product multiple times, mixing single and double strand roles could lead to a conflict.

For instance, a greedy direct mapping would not work for a reaction $B + B \rightarrow C$, since species B cannot be both a single and double strand. Therefore, to maintain universality, Soloveichik's transformation explicitly associates single strands to formal species.

9.1.2 Calculating Displacement Rates

So far, we covered the transformation of CRN's species and reactions to DNA strand displacement primitives, but for convenience, we omitted the specification of new rate constants q and q_{max} of the displacement reactions. In this section we will finalize the construction by adding rate constants.

First, we set the constant C_{max}, the maximal concentration, and assume that fuel species G_i, T_i for unimolecular reactions and L_i, B_i, T_i for bimolecular reactions, consumed and turned into waste, are provided at the concentration C_{max}, which is sufficiently higher than the concentrations of the signals. A formal proof showing the kinetic equivalence between a CRN and its transformed DNA strand displacement circuit can be found in the supplementary material of [140]. In the following, we provide rate constant calculations without proofs.

Let $\mathscr{B}$ be a set of all bimolecular reactions, $\mathscr{U}$ be a set of all unimolecular reactions and $\{X_i\}$ be a set of the CRN's species. We define the formal parameter σ_j for species X_j as

$$\sigma_j = \sum_{i \in \mathscr{B}|r_{i,1}=j} k_i$$

$$\sigma = \max_j \{\sigma_j\},$$

(9.3)

where $r_{i,1} = j$ holds if the species X_j is the first reactant in a bimolecular reaction i and σ is the maximal value for all species.

The maximal rate $q_{\max}$ at which most of the displacement reactions operate (Equations 9.1 and 9.1) is

$$q_{\max} = \max\{\max_{i \in \mathscr{U}} \frac{k_i}{C_{max}} + \sigma, \max_{i \in \mathscr{B}} k_i + \sigma, \max_{j}(2\sigma - \sigma_j)\} \qquad (9.4)$$

The buffering scaling factor γ, indicating the scale-up to rate constants necessary to cancel the effect of buffering of the signal species is

$$\gamma = \frac{(q_{\max} - \sigma)}{q_{\max}}. \qquad (9.5)$$

Note that so-called buffering must be introduced for species that are not sufficiently buffered by the first displacement of the bimolecular reactions. Formally, for every species X_j for which $\sigma_j < \sigma$, we formulate the buffering reaction illustrated in Figure 9.8 as

$$X_j + LS_j \xvar{\overset{qs_j}{\underset{q_{\max}}{\longleftrightarrow}}} HS_j + BS_j, \qquad (9.6)$$
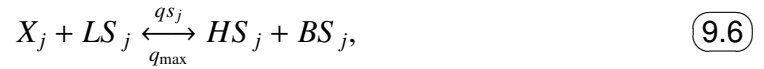
where fuel species LS_j and BS_j are provided at the concentration $C_{\max}$ and the rate constant $qs_j = \gamma^{-1}(\sigma - \sigma_j)$. The buffering reactions serve no purpose other than to increase the buffering load of DNA species that are not already maximally buffered, so that in the end all DNA species are buffered equally with accurate kinetics.

Finally, the new rate constant of a displacement reaction $S + G_i \xrightarrow{q_i} W_{1,i} + O_i$ mapped
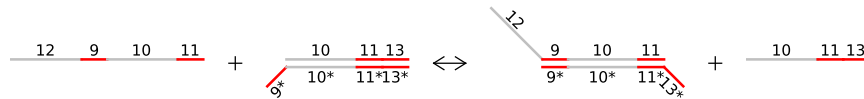
Figure 9.8: Example of a buffering reaction, a bidirectional displacement, of species X_j, defined as $X_j + LS_j \longleftrightarrow HS_j + BS_j$. Long domains are gray and short domains are red.

from a unimolecular reaction $S \xrightarrow{k_i} c_1 P_1 + \ldots + c_n P_n$ is

$$q_i = \gamma^{-1} k_i C_{\max}^{-1} \qquad (9.7)$$

and for a displacement reaction $S_1 + L \underset{q_{\max}}{\overset{q_i}{\longleftrightarrow}} H_i + B_i$ mapped from a bimolecular reaction $S_1 + S_2 \xrightarrow{k_i} c_1 P_1 + \ldots + c_n P_n$

$$q_i = \gamma^{-1} k_i. \qquad (9.8)$$

To replicate the displacement reaction rates the toehold domains that actively partic-ipate in a displacement must be sequenced such that their binding strength matches the prescribed rate.

9.1.3 DNA Strand Implementation of Our Models

We implemented and validated Soloveichik's compilation algorithm as a module in the COEL simulation framework (Chapter 10). Besides converting reactions, rates, and intro-ducing new symbolic species, our implementation generated the full domain specification of the DNA strands using Visual DSD syntax [91]. A DNA strand parser encoded by this syntax with a visualization mechanism was also implemented and integrated into COEL. Although we used a slightly different numbering of the domains, the correctness of the overall construction holds.

We applied our implementation of Soloveichik's compilation and embedded visualiza-tion to two models—the linear chemical perceptron and the manual delay line. Although the transformation could be applied to any of our models, we chose the linear chemical perceptron introduced in Section 8.4.2 because of its simplicity, analytical origin, and

equivalence to a linear neural network perceptron. To demonstrate the DNA compilation on a CRN model other than a chemical learner, we chose a manual signalling delay line, which is small and regular, and therefore better fitted than the most advanced and best-performing parallel-accessible delay line.

To be as close to physical reality as possible we set the maximal concentration $C_{max} =$ $10^{-5}M$, typical for DNA strand displacement reactions, assuming the fuel species are injected as $[G_i] = [T_i] = [L_i] = [B_i] = C_{max}$ for all reactions and $[LS_j] = [BS_j] = C_{max}$ for all species. Instead of using the formal rates of the original CRNs we rescale the timing of our two systems such that the maximal rate constant of the compiled DNA strand displacement reactions is $q_{max} = 10^6 M^{-1} s^{-1}$, recommended by Soloveichik as a target speed achievable in a wet implementation of strand displacement. Since q_{max} is calculated by Equation 9.4 from a given set of reactions, we reverse-engineer scaling constants to fit the maximal rate after the compilation to $10^6 M^{-1} s^{-1}$.

Table 9.1: The species type counts for the DNA strand displacement implementation of the linear chemical perceptron with two inputs and the manual signalling delay line of size three. The signals (sigs) are the original CRN species. The total number of the DNA species (strands) is 217 for the linear perceptron and 49 for the manual signalling delay line. The number of species that need to be actively provided to the system, i.e., the input signals and the fuel strands G, T, L, B, BS, and LS, is 106 (LP) and 22 (MDL). The rest are either intermediates (O, H, and HS), produced and consumed in a cascade, or wastes ($WT1$ and $WT2$).

(a) LP

Type	In Sigs	Other Sigs	G	T	L	B	BS	LS	O	H	HS	$WT1$	$WT2$	Total
#	9	12	5	18	17	17	20	20	22	17	20	22	18	217

(b) MDL

Type	In Sigs	Other Sigs	G	T	L	B	BS	LS	O	H	HS	$WT1$	$WT2$	Total
#	4	6	3	3	3	3	3	3	6	3	3	6	3	49

Table 9.2: The reactions of the linear chemical perceptron with two inputs and the manual signalling delay line of size three with the original and scaled rates (unimolecular in s^{-1} and bimolecular in $M^{-1}s^{-1}$).

(a) LP

Reaction	Rate	Scaled Rate
$S_{in} \rightarrow Y^{\ominus} + S_{in}^L$	1	0.004
$S_{in} + W_0 \rightarrow Y + S_{in}^L + W_0$	1	83333.33
$X_1 \rightarrow Y^{\ominus} + X_1^L$	1	0.004
$X_1 + W_1 \rightarrow Y + X_1^L + W_1$	1	83333.33
$X_2 \rightarrow Y^{\ominus} + X_2^L$	1	0.004
$X_2 + W_2 \rightarrow Y + X_2^L + W_2$	1	83333.33
$Y + Y^{\ominus} \rightarrow \lambda$	5	416666.67
$\hat{Y} + S_L \rightarrow E^{\oplus} + S_L$	1	83333.33
$Y + S_L \rightarrow E^{\ominus} + S_L$	1	83333.33
$E^{\oplus} + E_{decay} \rightarrow E_{decay}$	1	83333.33
$E^{\ominus} + E_{decay} \rightarrow E_{decay}$	1	83333.33
$E^{\oplus} \rightarrow W^{\oplus} + E^{\oplus}$	1	0.004
$E^{\ominus} \rightarrow W^{\ominus} + E^{\ominus}$	1	0.004
$W^{\oplus} + S_{in}^L \rightarrow W_0 + S_{in}^L$	1	83333.33
$W^{\ominus} + S_{in}^L \rightarrow W_0^{\ominus} + S_{in}^L$	1	83333.33
$W_0 + W_0^{\ominus} \rightarrow \lambda$	5	416666.67
$W^{\oplus} + X_1^L \rightarrow W_1 + X_1^L$	1	83333.33
$W^{\ominus} + X_1^L \rightarrow W_1^{\ominus} + X_1^L$	1	83333.33
$W_1 + W_1^{\ominus} \rightarrow \lambda$	5	416666.67
$W^{\oplus} + X_2^L \rightarrow W_2 + X_2^L$	1	83333.33
$W^{\ominus} + X_2^L \rightarrow W_2^{\ominus} + X_2^L$	1	83333.33
$W_2 + W_2^{\ominus} \rightarrow \lambda$	5	416666.67

(b) MDL

Reaction	Rate	Scaled Rate
$X + X_1^S \rightarrow X_1 + X_1^C + X_1^S$	5.0	500000
$X_1^C + X_2^S \rightarrow X_2 + X_2^C + X_2^S$	5.0	500000
$X_2^C + X_3^S \rightarrow X_3 + X_3^C + X_3^S$	5.0	500000
$X_1^S \rightarrow \lambda$	0.5	0.002
$X_2^S \rightarrow \lambda$	0.5	0.002
$X_3^S \rightarrow \lambda$	0.5	0.002

Linear Chemical Perceptron Implementation

As showed analytically, the rate constants of the linear chemical perceptron are not important and the system could work with arbitrarily picked constants that preserve inherent symmetries of the system such as treating the weights W_0, W_1, W_2 equally. Even with broken symmetry the system could perform fairly well, but then our proofs and the derived closed formulas would not hold. The only rate constants that we set explicitly were of the input-weight integration reactions. Recall that we set these rates to 1 to move the zero-weight value to the concentration of 1. The ODE-version of a linear chemical perceptron,

as we tested it, has all rate constants equal to 1 besides the annihilatory reactions. Since the annihilatory reactions process terminal species, they could be arbitrary but for faster execution we set them to 5.

Now we need to adjust the rates such that after the compilation $q_{max} = 10^6 M^{-1} s^{-1}$. We achieve that by scaling all unimolecular reactions by a factor $1/\alpha$ and all biomolecular reactions by a factor $1/\beta$ where $\alpha = 250$ and $\beta = \frac{6}{5} 10^{-6}$ shown in Table 9.2(a). Also, note that we moved from a unitless CRN to a CRN scaled to operate in the maximal concentration of $10\mu M$, where the signal species are expected to be provided at nM concentrations. The rates are scaled to s^{-1} for unimolecular and $M^{-1} s^{-1}$ for bimolecular reactions.

Because of many intermediate reactions and species, the original linear chemical perceptron model with 22 reactions and 21 species compiled to 77 displacement reactions and 217 DNA strand species (Table 9.1(a)) consisting of 104 domains (with complements). Out of all displacement reactions, 20 served buffering. Table 9.3 shows all displacements obtained by converting all unimolecular and bimolecular reactions. Table 9.4 shows the buffering reactions for the species with $\sigma_i < \sigma$. The key calculated paramterers used for the displacement rates are $\sigma = 5 \times 10^5$ and $\gamma^{-1} = 2$. Also, to avoid confusion with the weight species W_i, we renamed the wastes $W1$ and $W2$ to $WT1$ and $WT2$.

Manual Signalling Delay Implementation

In this section we present a DNA strand displacement implemention of the manual signalling delay line of size three showed in Figure 6.2. The rate constants of the copy reactions are set to 5 and the decays of the copy signals to 0.5, assuming the time frame to produce the cached values is within 20 time steps. Similarly to the linear chemical perceptron, we adjust the rates such that after the compilation $q_{max} = 10^6 M^{-1} s^{-1}$. We achieve that by scaling all unimolecular reactions by a factor $1/\alpha$ and all biomolecular

Table 9.3: The original and compiled DNA strand displacement reactions of the linear chemical perceptron with the forward and reverse rates ($M^{-1}s^{-1}$).

Reaction	DNA-SD Reactions	Forward Rate	Reverse Rate
$S_{in} \rightarrow Y^{\ominus} + S_{in}^{L}$	$S_{in} + G_{R01} \rightarrow WT1_{R01} + O_{R01}$	800	
	$O_{R01} + T_{R01} \rightarrow WT2_{R01} + Y^{\ominus} + S_{in}^{L}$	10^6	
$S_{in} + W_0 \rightarrow Y + S_{in}^{L} + W_0$	$S_{in} + L_{R02} \leftrightarrow H_{R02} + B_{R02}$	166666.66	10^6
	$W_0 + H_{R02} \rightarrow WT1_{R02} + O_{R02}$	10^6	
	$O_{R02} + T_{R02} \rightarrow WT2_{R02} + Y + S_{in}^{L} + W_0$	10^6	
$X_1 \rightarrow Y^{\ominus} + X_1^{L}$	$X_1 + G_{R03} \rightarrow WT1_{R03} + O_{R03}$	800	
	$O_{R03} + T_{R03} \rightarrow WT2_{R03} + Y^{\ominus} + X_1^{L}$	10^6	
$X_1 + W_1 \rightarrow Y + X_1^{L} + W_1$	$X_1 + L_{R04} \leftrightarrow H_{R04} + B_{R04}$	166666.66	10^6
	$W_1 + H_{R04} \rightarrow WT1_{R04} + O_{R04}$	10^6	
	$O_{R04} + T_{R04} \rightarrow WT2_{R04} + Y + X_1^{L} + W_1$	10^6	
$X_2 \rightarrow Y^{\ominus} + X_2^{L}$	$X_2 + G_{R05} \rightarrow WT1_{R05} + O_{R05}$	800	
	$O_{R05} + T_{R05} \rightarrow WT2_{R05} + Y^{\ominus} + X_2^{L}$	10^6	
$X_2 + W_2 \rightarrow Y + X_2^{L} + W_2$	$X_2 + L_{R06} \leftrightarrow H_{R06} + B_{R06}$	166666.66	10^6
	$W_2 + H_{R06} \rightarrow WT1_{R06} + O_{R06}$	10^6	
	$O_{R06} + T_{R06} \rightarrow WT2_{R06} + Y + X_2^{L} + W_2$	10^6	
$Y + Y^{\ominus} \rightarrow \lambda$	$Y + L_{R07} \leftrightarrow H_{R07} + B_{R07}$	833333.33	10^6
	$Y^{\ominus} + H_{R07} \rightarrow WT1_{R07} + O_{R07}$	10^6	
$\hat{Y} + S_L \rightarrow E^{\oplus} + S_L$	$\hat{Y} + L_{R08} \leftrightarrow H_{R08} + B_{R08}$	166666.66	10^6
	$S_L + H_{R08} \rightarrow WT1_{R08} + O_{R08}$	10^6	
	$O_{R08} + T_{R08} \rightarrow WT2_{R08} + E^{\oplus} + S_L$	10^6	
$Y + S_L \rightarrow E^{\ominus} + S_L$	$Y + L_{R09} \leftrightarrow H_{R09} + B_{R09}$	166666.66	10^6
	$S_L + H_{R09} \rightarrow WT1_{R09} + O_{R09}$	10^6	
	$O_{R09} + T_{R09} \rightarrow WT2_{R09} + E^{\ominus} + S_L$	10^6	
$E^{\oplus} + E_{decay} \rightarrow E_{decay}$	$E^{\oplus} + L_{R10} \leftrightarrow H_{R10} + B_{R10}$	166666.66	10^6
	$E_{decay} + H_{R10} \rightarrow WT1_{R10} + O_{R10}$	10^6	
	$O_{R10} + T_{R10} \rightarrow WT2_{R10} + E_{decay}$	10^6	
$E^{\ominus} + E_{decay} \rightarrow E_{decay}$	$E^{\ominus} + L_{R11} \leftrightarrow H_{R11} + B_{R11}$	166666.66	10^6
	$E_{decay} + H_{R11} \rightarrow WT1_{R11} + O_{R11}$	10^6	
	$O_{R11} + T_{R11} \rightarrow WT2_{R11} + E_{decay}$	10^6	
$E^{\oplus} \rightarrow W^{\oplus} + E^{\oplus}$	$E^{\oplus} + G_{R12} \rightarrow WT1_{R12} + O_{R12}$	800	
	$O_{R12} + T_{R12} \rightarrow WT2_{R12} + W^{\oplus} + E^{\oplus}$	10^6	
$E^{\ominus} \rightarrow W^{\ominus} + E^{\ominus}$	$E^{\ominus} + G_{R13} \rightarrow WT1_{R13} + O_{R13}$	800	
	$O_{R13} + T_{R13} \rightarrow WT2_{R13} + W^{\ominus} + E^{\ominus}$	10^6	
$W^{\oplus} + S_{in}^{L} \rightarrow W_0 + S_{in}^{L}$	$W^{\oplus} + L_{R14} \leftrightarrow H_{R14} + B_{R14}$	166666.66	10^6
	$S_{in}^{L} + H_{R14} \rightarrow WT1_{R14} + O_{R14}$	10^6	
	$O_{R14} + T_{R14} \rightarrow WT2_{R14} + W_0 + S_{in}^{L}$	10^6	
$W^{\ominus} + S_{in}^{L} \rightarrow W_0^{\ominus} + S_{in}^{L}$	$W^{\ominus} + L_{R15} \leftrightarrow H_{R15} + B_{R15}$	166666.66	10^6
	$S_{in}^{L} + H_{R15} \rightarrow WT1_{R15} + O_{R15}$	10^6	
	$O_{R15} + T_{R15} \rightarrow WT2_{R15} + W_0^{\ominus} + S_{in}^{L}$	10^6	
$W_0 + W_0^{\ominus} \rightarrow \lambda$	$W_0 + L_{R16} \leftrightarrow H_{R16} + B_{R16}$	833333.33	10^6
	$W_0^{\ominus} + H_{R16} \rightarrow WT1_{R16} + O_{R16}$	10^6	

$W^\oplus + X_1^L \to W_1 + X_1^L$	$W^\oplus + L_{R17} \leftrightarrow H_{R17} + B_{R17}$	166666.66	10^6
	$X_1^L + H_{R17} \to WT1_{R17} + O_{R17}$	10^6	
	$O_{R17} + T_{R17} \to WT2_{R17} + W_1 + X_1^L$	10^6	
$W^\ominus + X_1^L \to W_1^\ominus + X_1^L$	$W^\ominus + L_{R18} \leftrightarrow H_{R18} + B_{R18}$	166666.66	10^6
	$X_1^L + H_{R18} \to WT1_{R18} + O_{R18}$	10^6	
	$O_{R18} + T_{R18} \to WT2_{R18} + W_1^\ominus + X_1^L$	10^6	
$W_1 + W_1^\ominus \to \lambda$	$W_1 + L_{R19} \leftrightarrow H_{R19} + B_{R19}$	833333.33	10^6
	$W_1^\ominus + H_{R19} \to WT1_{R19} + O_{R19}$	10^6	
$W^\oplus + X_2^L \to W_2 + X_2^L$	$W^\oplus + L_{R20} \leftrightarrow H_{R20} + B_{R20}$	166666.66	10^6
	$X_2^L + H_{R20} \to WT1_{R20} + O_{R20}$	10^6	
	$O_{R20} + T_{R20} \to WT2_{R20} + W_2 + X_2^L$	10^6	
$W^\ominus + X_2^L \to W_2^\ominus + X_2^L$	$W^\ominus + L_{R21} \leftrightarrow H_{R21} + B_{R21}$	166666.66	10^6
	$X_2^L + H_{R21} \to WT1_{R21} + O_{R21}$	10^6	
	$O_{R21} + T_{R21} \to WT2_{R21} + W_2^\ominus + X_2^L$	10^6	
$W_2 + W_2^\ominus \to \lambda$	$W_2 + L_{R22} \leftrightarrow H_{R22} + B_{R22}$	833333.33	10^6
	$W_2^\ominus + H_{R22} \to WT1_{R22} + O_{R22}$	10^6	

Table 9.4: The buffering reactions of the DNA-strand implemented linear chemical perceptron with the forward and reverse rates in $M^{-1}s^{-1}$.

Species	Buffering DNA-SD Reaction	Forward Rate	Reverse Rate
S_{in}	$S_{in} + LS_{S_{in}} \leftrightarrow HS_{S_{in}} + BS_{S_{in}}$	833333.33	10^6
X_1	$X_1 + LS_{X_1} \leftrightarrow HS_{X_1} + BS_{X_1}$	833333.33	10^6
X_2	$X_2 + LS_{X_2} \leftrightarrow HS_{X_2} + BS_{X_2}$	833333.33	10^6
W_0	$W_0 + LS_{W_0} \leftrightarrow HS_{W_0} + BS_{W_0}$	166666.66	10^6
W_1	$W_1 + LS_{W_1} \leftrightarrow HS_{W_1} + BS_{W_1}$	166666.66	10^6
W_2	$W_2 + LS_{W_2} \leftrightarrow HS_{W_2} + BS_{W_2}$	166666.66	10^6
S_{in}^L	$S_{in}^L + LS_{S_{in}^L} \leftrightarrow HS_{S_{in}^L} + BS_{S_{in}^L}$	10^6	10^6
X_1^L	$X_1^L + LS_{X_1^L} \leftrightarrow HS_{X_1^L} + BS_{X_1^L}$	10^6	10^6
X_2^L	$X_2^L + LS_{X_2^L} \leftrightarrow HS_{X_2^L} + BS_{X_2^L}$	10^6	10^6
$E^\ominus$	$E^\ominus + LS_{E^\ominus} \leftrightarrow HS_{E^\ominus} + BS_{E^\ominus}$	833333.33	10^6
$E^\oplus$	$E^\oplus + LS_{E^\oplus} \leftrightarrow HS_{E^\oplus} + BS_{E^\oplus}$	833333.33	10^6
$W^\ominus$	$W^\ominus + LS_{W^\ominus} \leftrightarrow HS_{W^\ominus} + BS_{W^\ominus}$	500000	10^6
$W^\oplus$	$W^\oplus + LS_{W^\oplus} \leftrightarrow HS_{W^\oplus} + BS_{W^\oplus}$	500000	10^6
$W_0^\ominus$	$W_0^\ominus + LS_{W0^\ominus} \leftrightarrow HS_{W0^\ominus} + BS_{W_0^\ominus}$	10^6	10^6
$W_1^\ominus$	$W_1^\ominus + LS_{W_1^\ominus} \leftrightarrow HS_{W_1^\ominus} + BS_{W_1^\ominus}$	10^6	10^6
$W_2^\ominus$	$W_2^\ominus + LS_{W_2^\ominus} \leftrightarrow HS_{W_2^\ominus} + BS_{W_2^\ominus}$	10^6	10^6
S_L	$S_L + LS_{S_L} \leftrightarrow HS_{S_L} + BS_{S_L}$	10^6	10^6
$Y^\ominus$	$Y^\ominus + LS_{Y^\ominus} \leftrightarrow HS_{Y^\ominus} + BS_{Y^\ominus}$	10^6	10^6
$\hat{Y}$	$\hat{Y} + LS_{\hat{Y}} \leftrightarrow HS_{\hat{Y}} + BS_{\hat{Y}}$	833333.33	10^6
E_{decay}	$E_{decay} + LS_{E_{decay}} \leftrightarrow HS_{E_{decay}} + BS_{E_{decay}}$	10^6	10^6

reactions by a factor $1/\beta$, where $\alpha = 250$ and $\beta = 10^{-5}$, as shown in Table 9.2(b).

The original manual signalling delay line with 6 reactions and 10 species compiles to 15 displacement reactions and 49 DNA strand species (Table 9.1(b)), consisting of 43 domains (with complements). Out of all displacement reactions 3 served buffering. Table 9.5 shows the displacements obtained by converting all unimolecular and bimolecular reactions and Table 9.6 shows the buffering reactions added for accuracy for the species with $\sigma_i < \sigma$. As before, the calculated variables used for the displacement rates are $\sigma = 5 \times 10^5$ and $\gamma^{-1} = 2$.

Table 9.5: The original and compiled DNA strand displacement reactions of the manual signalling delay line with the forward and reverse rates ($M^{-1}s^{-1}$).

Reaction	DNA-SD Reactions	Forward Rate	Reverse Rate
$X + X_1^S \rightarrow X_1 + X_1^C + X_1^S$	$X + L_{R01} \leftrightarrow H_{R01} + B_{R01}$	10^6	10^6
	$X_1^S + H_{R01} \rightarrow WT1_{R01} + O_{R01}$	10^6	
	$O_{R01} + T_{R01} \rightarrow WT2_{R01} + X_1 + X_1^C + X_1^S$	10^6	
$X_1^C + X_2^S \rightarrow X_2 + X_2^C + X_2^S$	$X_1^C + L_{R02} \leftrightarrow H_{R02} + B_{R02}$	10^6	10^6
	$X_2^S + H_{R02} \rightarrow WT1_{R02} + O_{R02}$	10^6	
	$O_{R02} + T_{R02} \rightarrow WT2_{R02} + X_2 + X_2^C + X_2^S$	10^6	
$X_2^C + X_3^S \rightarrow X_3 + X_3^C + X_3^S$	$X_2^C + L_{R03} \leftrightarrow H_{R03} + B_{R03}$	10^6	10^6
	$X_3^S + H_{R03} \rightarrow WT1_{R03} + O_{R03}$	10^6	
	$O_{R03} + T_{R03} \rightarrow WT2_{R03} + X_3 + X_3^C + X_3^S$	10^6	
$X_1^S \rightarrow \lambda$	$X_1^S + G_{R04} \rightarrow WT1_{R04} + O_{R04}$	400	
$X_2^S \rightarrow \lambda$	$X_2^S + G_{R05} \rightarrow WT1_{R05} + O_{R05}$	400	
$X_3^S \rightarrow \lambda$	$X_3^S + G_{R06} \rightarrow WT1_{R06} + O_{R06}$	400	

Table 9.6: The buffering reactions of the DNA-strand implemented manual delay line with the forward and reverse rates in $M^{-1}s^{-1}$.

Species	Buffering DNA-SD Reaction	Forward Rate	Reverse Rate
X_1^S	$X_1^S + LS_{X_1^S} \leftrightarrow HS_{X_1^S} + BS_{X_1^S}$	10^6	10^6
X_2^S	$X_2^S + LS_{X_2^S} \leftrightarrow HS_{X_2^S} + BS_{X_2^S}$	10^6	10^6
X_3^S	$X_3^S + LS_{X_3^S} \leftrightarrow HS_{X_3^S} + BS_{X_3^S}$	10^6	10^6

$$S_{in} + G_{R01} \rightarrow WT1_{R01} + O_{R01}$$

$$O_{R01} + T_{R01} \rightarrow WT2_{R01} + Y^{\ominus} + S_{in}^{L}$$

$$S_{in} + L_{R02} \leftrightarrow H_{R02} + B_{R02}$$

$$W_0 + H_{R02} \rightarrow WT1_{R02} + O_{R02}$$

$$O_{R02} + T_{R02} \rightarrow WT2_{R02} + Y + S_{in}^{L} + W_0$$

$$X_1 + G_{R03} \rightarrow WT1_{R03} + O_{R03}$$

$$O_{R03} + T_{R03} \rightarrow WT2_{R03} + Y^{\ominus} + X_{1}^{L}$$
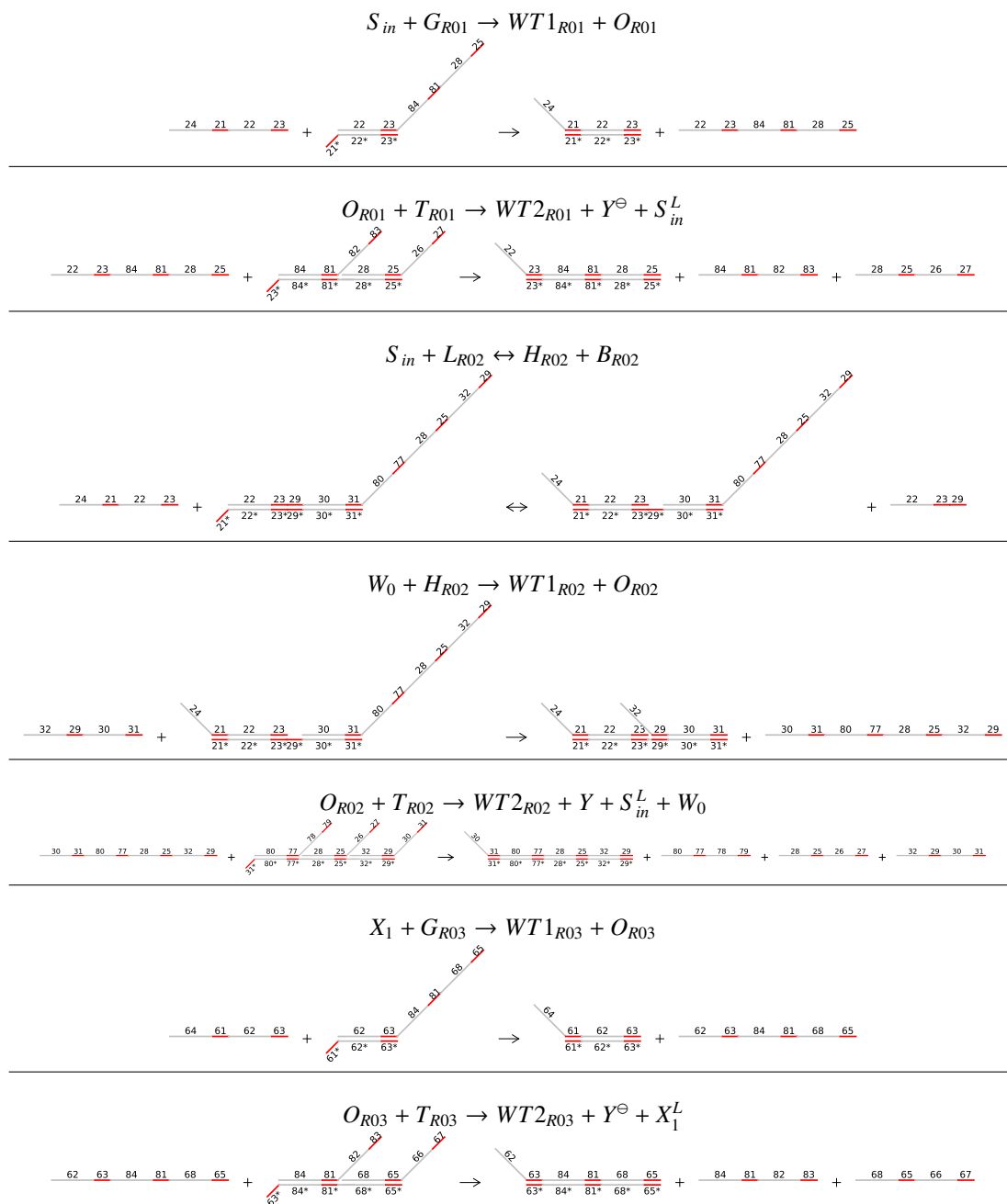
Figure 9.9: The domain-specified structures of the first 7 displacements of the linear chemical perceptron from Table 9.3. The full list is available in Appendix, Figure C.1.
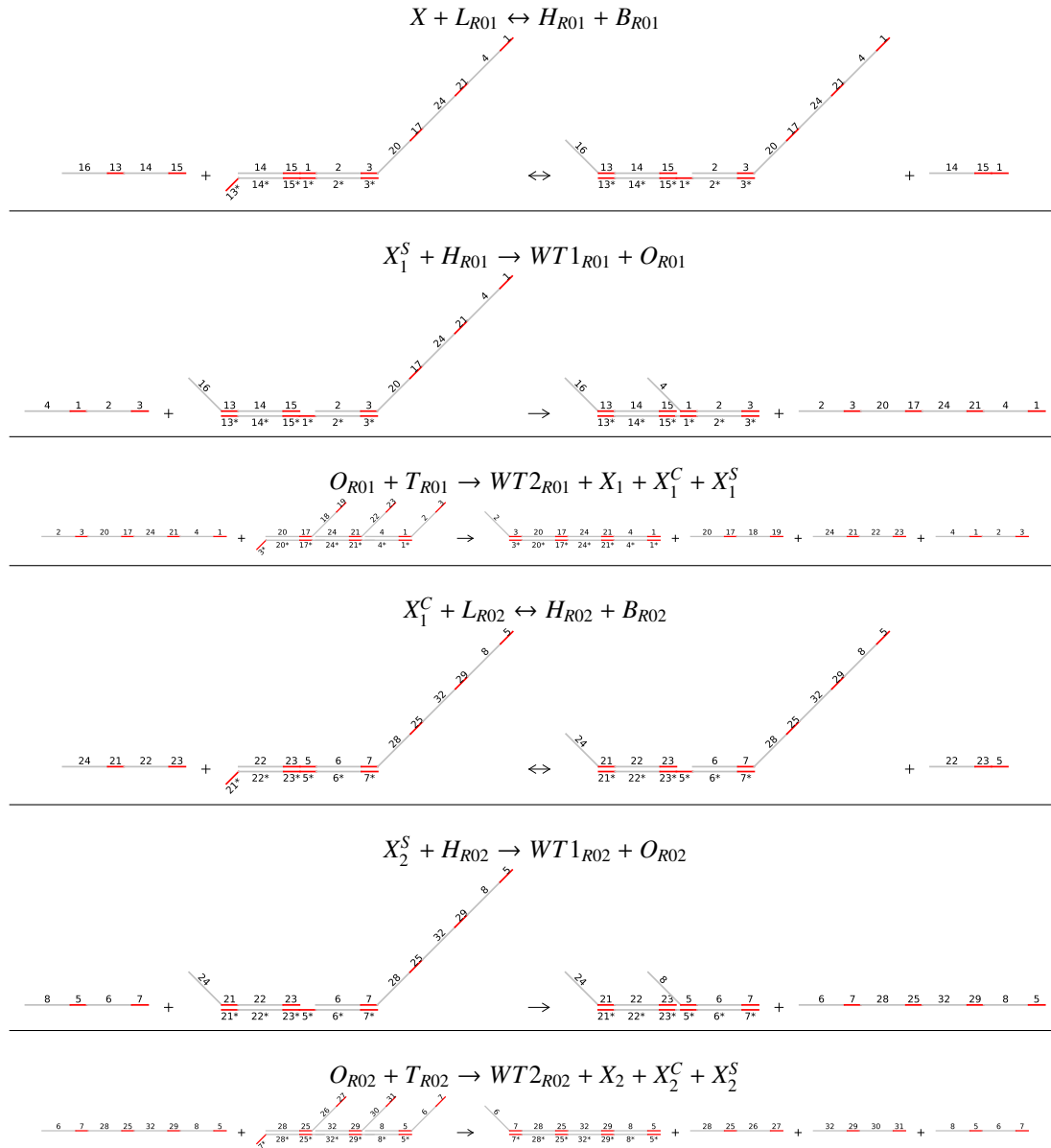
Figure 9.10: The domain-specified structures of the first 6 displacements of the manual signalling delay line from Table 9.5. The full list is available in Appendix, Figure C.2.

9.1.4 Discussion

Once the DNA strands of our models are sequenced and synthesized, they could directly lead to biochemical implementations. Note that only the signal and fuel species, which are to be provided to the system during an experiment, need to be synthesized. The number of such species is 106 for the linear chemical perceptron and 22 for the manual signalling delay line. The remaining species are either intermediates or wastes. A state-of-the-art DNA strand displacement system to calculate square root [127], experimentally constructed and verified in a lab, consists of 130 DNA strands. Even though the so-called see-saw gates used in the experiment are simpler than the ones produced by Soloveichik's transformation, the size of both our DNA strand systems is within the same complexity range. Thus, taking into count practical considerations and cost, they are in principal wet-implementable.

The displacement reaction rates are determined by the binding strength of the toehold domains. Even though in practice a limited number of exact rate constants are achievable, they are distributed over many orders of magnitude [140]. As we mentioned, both our example CRNs, and therefore also their DNA strand displacement circuits, are insensitive to a specific setting of rate constants. We expect that a failure to precisely replicate the displacement rates under experimental conditions would be absorbed by the systems' robustness and would practically affect only the waiting times of the systems' stages.

An inherent challenge of DNA strand displacement is to minimize unwanted binding of non-complementary domains. Even though intelligent sequence designers (e.g., NUPACK) address that, for large number of domains, such as as in the linear chemical perceptron, partial crosstalk and leakage is inevitable. We can expect that the leakage of even larger-scale DNA strand systems will further decline [151].

Furthermore, the fuel species expected to be provided at large concentrations $10^{-5}M$ are depleted and turned into inactive waste as the system evolves. As the fuel is consumed, the system becomes less accurate and departs from its expected kinetics. One-shot-only systems are naturally resistant to that, however, any chemical learner or delay line is expected to work repeatedly in many iterations throughout its lifecycle. Since the signal species are provided at nM as opposed to $10\mu M$ (fuel strands), it might take hundreds of iterations until the accuracy starts to decline. On the other hand, if we opt for true reusability of the linear chemical perceptron and the manual signalling delay line during training, we would need to continually add new fuel gates to the solution. In order to preserve the kinetics, the correct amount of replacement structures would need to be added repetitively to counter balance their consumption. That kind of experimental setup would be nontrivial to reproduce.

Besides the Soloveichik's DNA strand displacement transformation there are two more DNA strand implementation methods proposed by Luca Cardelli. The building blocks are known as "three domain" [34] and "two domain" strands and gates [35]. These use restricted classes of DNA strands to denote the chemical species (with two or three domains respectively). The main difference compared to Soloveichik's method is for experimental implementations. In particular, the three-domain and even moreso the two-domain scheme uses very simple structures for the gates and strands. They are thus easier to synthesize, but have more transformation steps, so they are slower to emulate a given set of reactions.

9.2 DEOXYRIBOZYME DNA

Besides ordinary reactions $A+B \leftrightarrow C+D$ covered by DNA strand displacement, DNA also serves enzymatic (catalytic) reactions. DNA enzymes called deoxyribozymes [30,95,145] catalyze the transformation of oligonucleotides (short, single-stranded DNAs). They were applied, e.g., for modeling logic gates [100, 144], a tic-tac-toe automaton [121, 145], and a half-adder [146].

A deoxyribozyme consists of a stem loop called a catalytic core and two short toehold "legs" (sticky ends). In a core deoxyribozyme reaction as shown in Figure 9.11, a deoxyribozyme D binds to a fluorogenic substrate, oligonucleotide Q-F, consisting of domains 1 and 2 complementary to the deoxyribozyme's legs 1^* and 2^*. The deoxyribozyme gate D then cleaves the attached oligonucleotide Q-F at the "weak" site of a single ribonucleotide (rA), which is used to ensure an expected cleavage site. After the cleavage the single toehold bindings become weak and they detach from the complex as two products Q and F. Since the deoxyribozyme exits the reaction unaltered, it acts as a catalyst and we can write the reaction as $D + Q$-$S \rightarrow D + Q + S$. The product F contains a chemical called fluorescein that emits fluorescence. It can easily be measured and interpreted as the output of the system.

In our implementations we are going to use the so-called NOT gates [100]. A NOT gate is a deoxyribozyme where a part of the stem-loop sequence is replaced by a specific strand i^*. Besides this structural change, a deoxyribozyme works normally and if presented, it cleaves a complementary oligonucleotide. However, as shown in Figure 9.12, if we provide a strand I, consisting of a domain i to the solution, it binds to the complementary loop, and distorts the shape of the catalytic core, thus inhibiting (disabling) the catalytic cleavage.

Note that many deoxyribozyme reaction types exist. For instance, in a *if-then* case, the substrate *S* of the upstream enzyme *U* deactivates the downstream enzyme *D*. If *S* is attached to *D* nothing happens, however, after the cleavage of *S* by *U*, *D* is activated, so the core reaction can go on. By combining different cascading types of upstream and downstream enzymes and the activation and deactivation relations, more complicated hierarchical scenarios could be served.
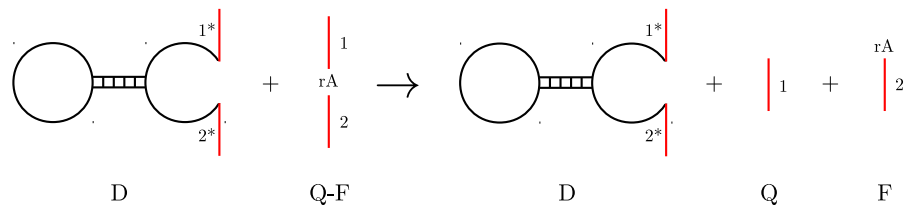
Figure 9.11: An example of catalytic DNA reaction $D + Q\text{-}F \rightarrow D + Q + F$: deoxyribozyme *D* cleaves an oligonucleotide *Q-F* into two parts *Q* and *F*.
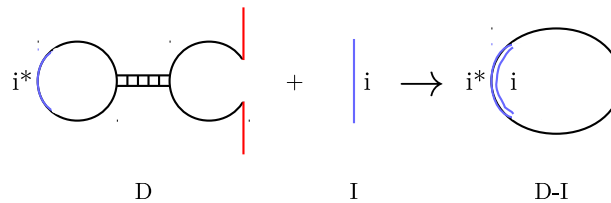
Figure 9.12: Deoxyribozyme-based NOT gate $D + I \rightarrow D\text{-}I$. A strand *i* binds to the stem loop sequence i^* of the enzyme *D* and deactivates it.

9.2.1 Deoxyribozyme-Based Implementation of Our Models

As opposed to DNA strand displacement, no automatic compilation technique to map a CRN to a deoxyribozyme-based circuit exists. The main reason why we opted for deoxyribozymes is higher reusability and serving of catalytic reactions. On the other hand, since we cannot rely on known compilation techniques the correctness of our constructions is purely qualitative. Our implementations are more speculative, but incorporate a

wider range of structural features than our generic, yet conservative DNA strand displacement implementations.

Since both our models heavily rely on catalysis, which is effectively served by deoxyribozymes, we could save a large amount of fuel gates and intermediates, eventually reducing the system size roughly to a third of what we presented previously.

In this section we provide a domain specification of oligonucleotides and deoxyribozymes carrying out the original CRN reactions. As we mentioned before, both our models are robust and work correctly with arbitrary rates obeying the inherent system symmetries. At this level of precision, we can simply assume some uniform rates reflected by the toehold binding strengths exist. The actual rates of wet implementations will affect only the timing of the system phases (e.g., input and desired output injection delay).

Manual Signalling Delay Implementation

In this section we present a deoxyribozyme-based implementation of the manual signalling delay line of size three, as shown in Figure 6.2. Because the core catalytic reactions of the delay line $X_i^C \xrightarrow{X_i^S} X_{i+1} + X_{i+1}^C$ are already in the right format for the standard cleavage reaction (Figure 9.11), we use a direct mapping and obtained three cascaded reactions shown in Figure 9.13. At each stage, a deoxyribozyme with legs k^* and $(k+1)^*$ cleaves the "top" part of the oligonucleotide, which was fed from the preceding phase. Hence the original oligo 1-2-3-4 representing the input species X continuously splits to X_1 with the toehold 1, X_2 represented by the toehold 2, and finally X_3 consisting of a single toehold 3. The problem with this rather naive scheme is that the oligo 1-2-3-4 contains the parts complementary to all the enzymes. As a matter of fact, there is nothing preventing the enzymes from a simultaneous (unwanted) cleavage of the original oligo once injected.

(a) $X_1^S + X \rightarrow X_1^S + X_1 + X_1^C$

(b) $X_2^S + X_1^C \rightarrow X_2^S + X_2 + X_2^C$

(c) $X_3^S + X_2^C \rightarrow X_3^S + X_3 + X_3^C$
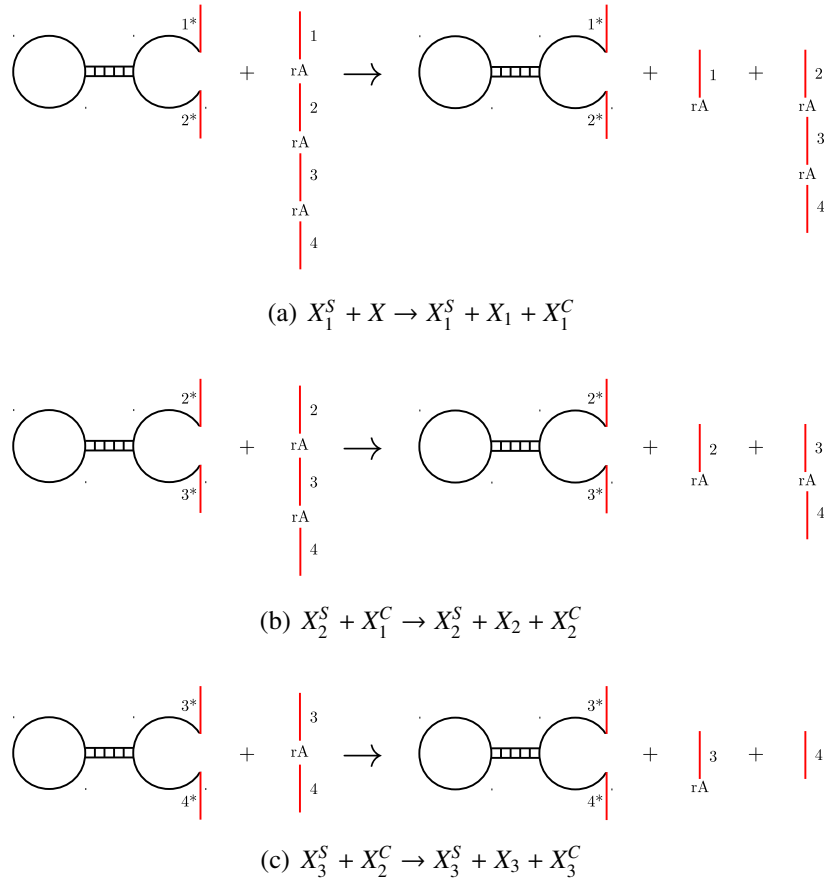
Figure 9.13: Naive deoxyribozyme-based implementation of the copy reactions of the manual signalling delay line of size three.

To fix the crosstalk issue of the naive mapping, we need a mechanism to transfer and protect cached species between the cleavage phases. We approach that with the help of DNA strand displacement. In particular, instead of directly producing the cached species, we introduce intermediates O_1 and O_2, which displace the actual X_1^C and X_2^C from the double-strand gates T_1 and T_2 producing wastes WT_1 and WT_2 as byproducts (Figure 9.14). In this scheme the unique enzymes' legs are ordered as $1^*, 2^*$; $3^*, 4^*$, and $5^*, 6^*$. The inpu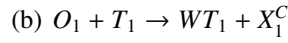t oligo X could be cleaved only by the deoxyribozyme X_1^S since the only pair of complementary toeholds that matches any deoxyribozyme is 1-2. This property induc-

tively holds for consecutive oligos X_1^C and X_2^C. Each oligo ends with the long domain e, which does not serve any purpose in catalyses and it only increases and stabilizes the DNA strand displacements. Note that for better circuit precision we can assume that the binding of the odd-indexed toeholds 1-1*, 3-3*, and 5-5* is stronger than of the even-indexed 2-2*, 4-4*, and 6-6*. This ensures that the oligos do not bind to the double-strands T_1 and T_2 before being cleaved. Without this assumption, the system would still work, yet more cumbersomely since it could still cleave the oligo attached to the lower strands of T_1 and T_2 because the toehold 1 and 2 respectively would be unmatched.

Even though we solved the problem of a simultaneous cleavage of the oligonucleotides, we unintentionally created another source of crosstalk, this time among the enzymes and the double strand (fuel) gates. More precisely, the deoxyribozyme X_2^S could potentially attach to the hanging toehold 3 of the gate T_1 (similarly for the enzyme X_3^S and the gate T_2). Since this unwanted binding is weak the overall correctness of the construction holds, however, it might result in a slower system's execution.

The final and also the most complex circuit prevents a disruption of the fuel gates by keeping the domains 3, 4 and 5, 6 fully attached to the lower strands. This requires a more clever domain numbering with the nested even-indexed domains and alternating the oligo ends. The original input X is now 1-2-6-e-4, where the long domain e is used as previously to increase potency of the strand displacements. At each level the gate T_i alternates its toehold position from left to right to protect the cleavage domains, which are similarly to the previous construction swapped. Another difference is that strand displacements are now bidirectional, e.g., the strand X_1^C could attach with its toehold 3 to the sticky end of the waste WT_1, however, once the oligos X_1^C and X_2^C are produced they are unidirectional cleaved by X_2^S and X_3^S respectively, so the reverse displacements are rather weak. This construction is easily generalizable to any number of cached values.

(a) $X_1^S + X \rightarrow X_1^S + X_1 + O_1$

(b) $O_1 + T_1 \rightarrow WT_1 + X_1^C$

(c) $X_2^S + X_1^C \rightarrow X_2^S + X_2 + O_2$

(d) $O_2 + T_2 \rightarrow WT_2 + X_2^C$

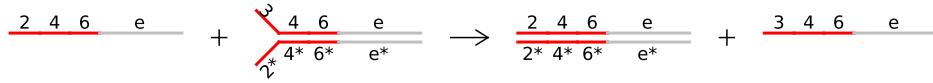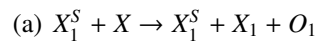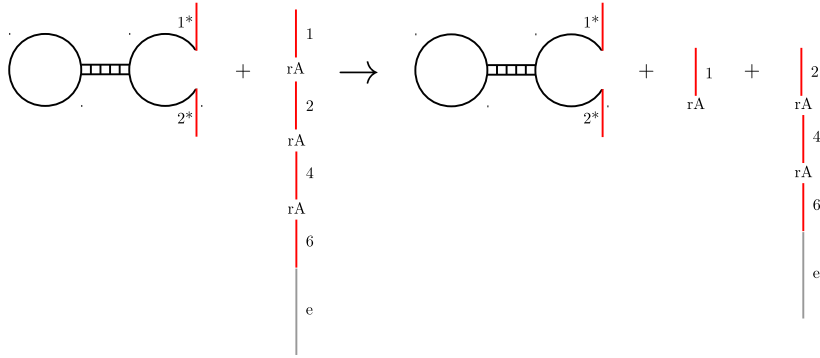(e) $X_3^S + X_2^C \rightarrow X_3^S + X_3 + X_3^C$
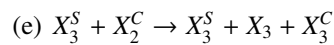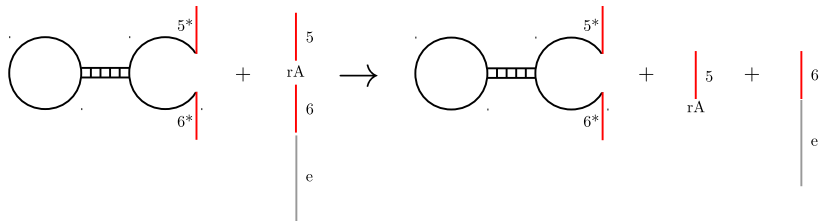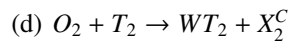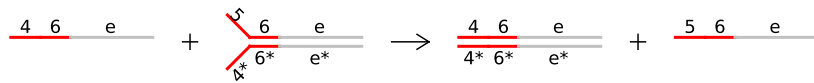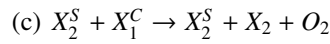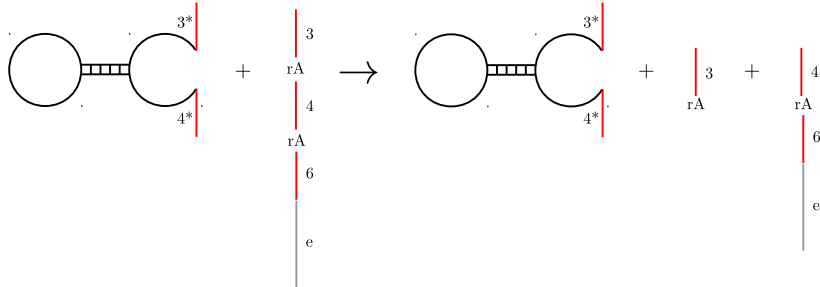
Figure 9.14: Deoxyribozyme-based implementation of the copy reactions of the manual signalling delay line of size three where an exchange of the cached species X_i^C between the stages is carried out by the DNA strand displacements.
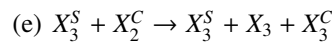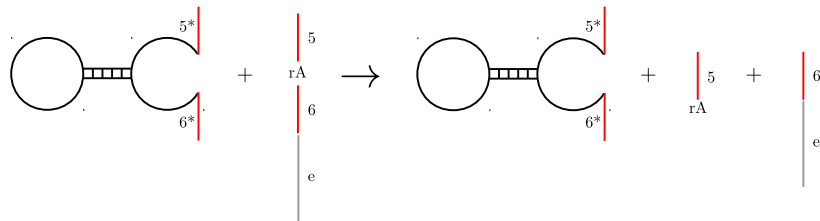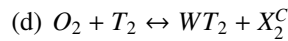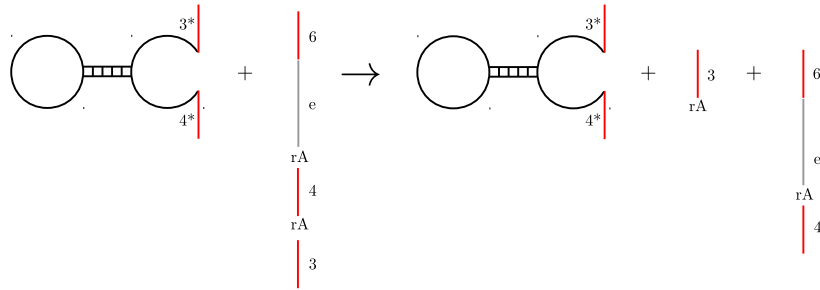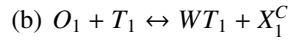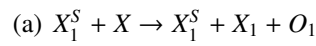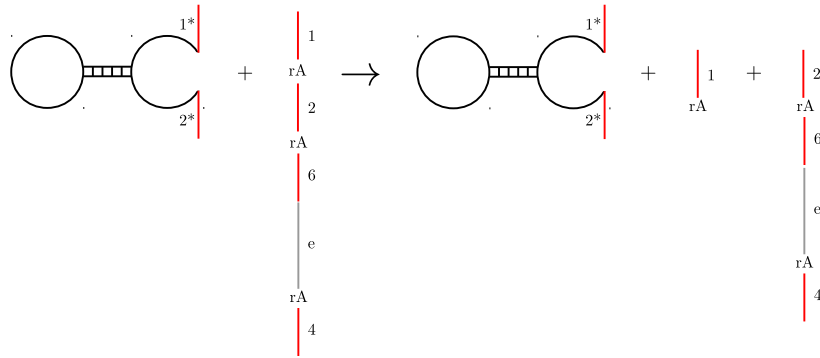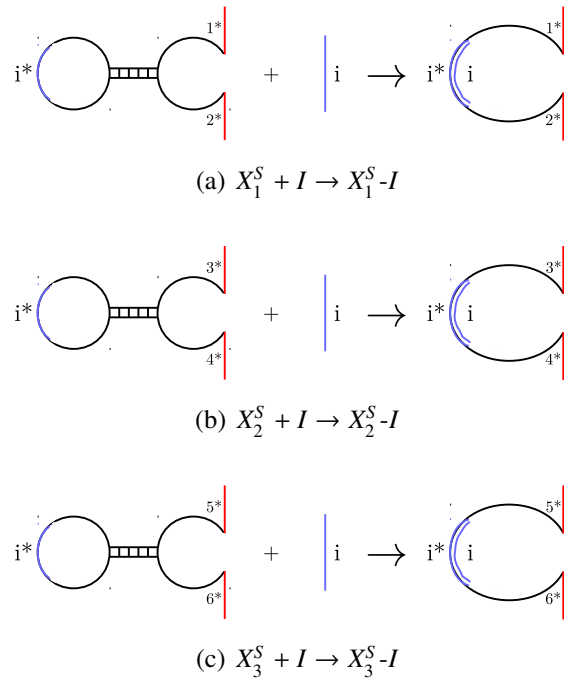
(a) $X_1^S + X \rightarrow X_1^S + X_1 + O_1$

(b) $O_1 + T_1 \leftrightarrow WT_1 + X_1^C$

(c) $X_2^S + X_1^C \rightarrow X_2^S + X_2 + O_2$

(d) $O_2 + T_2 \leftrightarrow WT_2 + X_2^C$

(e) $X_3^S + X_2^C \rightarrow X_3^S + X_3 + X_3^C$
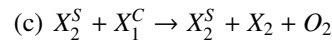
Figure 9.15: Final deoxyribozyme-based implementation of the copy reactions of the manual signalling delay line of size three where an exchange of the cached species X_i^C between the stages is carried out by the DNA strand displacements.

(a) $X_1^S + I \rightarrow X_1^S\text{-}I$

(b) $X_2^S + I \rightarrow X_2^S\text{-}I$

(c) $X_3^S + I \rightarrow X_3^S\text{-}I$

Figure 9.16: Deoxyribozyme-based implementation (NOT gates) of the signal decays of the manual signalling delay line of size three.

Similarly to how we handled the copy reactions of the manual signalling delay line, we need to implement the decays of the copy signal species. An easy solution is to assume the presence of a special deactivation species I, which is a long strand i implementing a NOT gate trigger for all three enzymes X_1^S, X_2^S, and X_3^S, whose stem loops contain a complementary sequence i^* (Figure 9.16). As a matter of fact, each enzyme, after it gets injected one by one in a backward order (as expected by the functioning of the delay line), starts cleaving its oligo, but as it proceeds, its activity slowly fades due to an interaction with the species I provided in abundance. Once it is fully deactivated we inject the signal for the higher level up to X_1^S.

Note that because synthesizing an oligo with multiple cleavage sites is expensive, we can continually add a new cleavage site after each stage (displacement). Then each oligo

X, X_1^C, and X_2^C would have only a single cleavage site and the intruding strands O_1 and O_2 would be plain strands (without any cleavage sites).

Overall, we transformed the original CRN model of the manual signalling delay line with 6 reactions to a deoxyribozyme-based circuit with 8 reactions: 3 cleavages, 3 gate deactivations (NOT gates), and 2 DNA strand displacements. The original set of 10 species expanded to 19 species consisting of 6 deoxyribozymes (3 regular and 3 NOT) and 13 oligos or single/double strands.

Linear Chemical Perceptron Implementation

In this section we present a deoxyribozyme-based implementation of the linear chemical perceptron with two inputs. Due to the size and the reaction complexity of this CRN, we focus only on mapping the input-weight integration reactions. More precisely, we implement the catalytic input-output reactions $X_i \xrightarrow{W_i} Y$ and $X_i \rightarrow Y^{\ominus}$ and the annihilation of the positive and negative outputs $Y + Y^{\ominus} \rightarrow \lambda$. Recall that the actual output is determined by the concentration of the weight species W_i. Here, for sake of simplicity, we removed the input-contribution species X_i^L produced along Y and $Y^{\ominus}$ because they do not serve the input-weight integration in any way and their final concentration is known to be equal to that of the original injected inputs.

The first step is to adjust the simple transformation $X_i \rightarrow Y^{\ominus}$ to a catalytic reaction $X_i \xrightarrow{W_i^{\ominus}} Y^{\ominus}$, where $W_i^{\ominus}$ is a new species (deoxyribozyme) with a constant concentration imposing a pressure at W_i, i.e., we assume that in the eventual learning part only W_i gets adapted. For generality we relabel $X_0 = S_{in}$. The main difficulty of the construction is to allow a dual race of two deoxyribozymes W_i and $W_i^{\ominus}$ over the single substrate X_i. Since the oligo X_i is shared, it needs to contain two cleavage sites, one for each weight. Further, we need to design it in a way such that the oligo X_i could be cleaved just once,
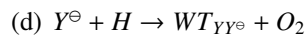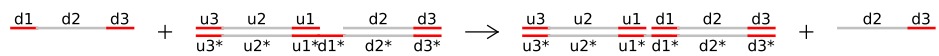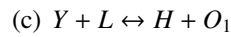
(a) $W_i + X_i \rightarrow W_i + Y + WT_i$

(b) $W_i^{\ominus} + X_i \rightarrow W_i^{\ominus} + Y^{\ominus} + WT_1^{\ominus}$

(c) $Y + L \leftrightarrow H + O_1$

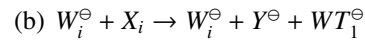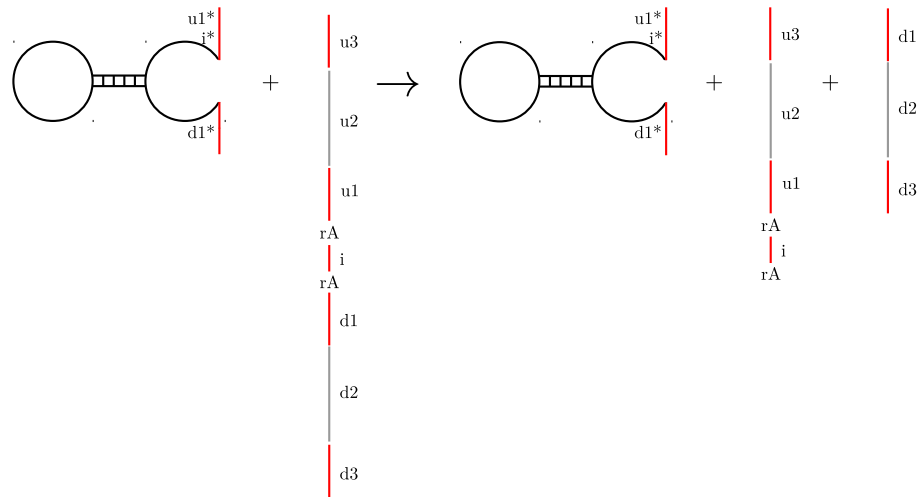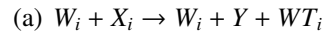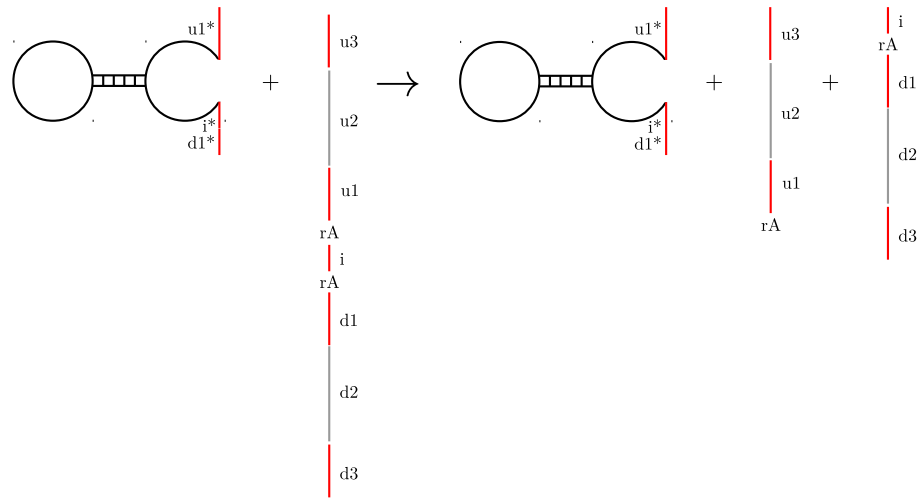(d) $Y^{\ominus} + H \rightarrow WT_{YY^{\ominus}} + O_2$
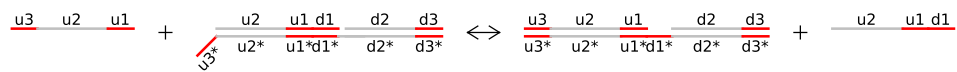
Figure 9.17: Deoxyribozyme-based implementation of the input-weight integration reactions of the linear chemical perceptron.

so if W_i cleaves it first, there must be a mechanism preventing $W_i^\ominus$ from further interference (and vice versa). We address this issue by a shared short toehold i wrapped around two cleavage sites. It is important to mention that i serves as a unique identifier of the oligo X_i preventing W_j and $W_j^\ominus$ where $i \neq j$ from cleaving. As shown in Figure 9.17, the deoxyribozyme W_i has two legs, $u1^*$ and i^*d1^*, cleaving the oligo X structured as $u3$-$u2$-$u1$-i-$d1$-$d2$-$d3$ (which we justify later), into the output Y and the waste WT_i consisting of the domains i-$d1$-$d2$-$d3$. Because of the overlap of the competing gates, the deoxyribozyme $W_i^\ominus$ with legs $u1^*i^*$ and $d1^*$ cannot fully bind to WT_i, hence WT_i becomes an inert waste. If $W_i^\ominus$ cleaves first, the output $Y^\ominus$ is produced instead. Whether more Y or $Y^\ominus$ will be produced depends on the concentrations of W_i and $W_i^\ominus$ and the binding strength of the toeholds $u1$, i, and $d1$, which we can assume is uniform. Note that instead of a linear structure we could use a circular oligo X_i introduced by Levy [93], keeping two cleavage sites.

Each pair of the weight races processes its own input and produces the outputs Y and $Y^\ominus$, shared across the input-weight integration reactions. The last part we need to cover is the annihilation of the regular output with the negative output $Y + Y^\ominus \rightarrow \lambda$. A straightforward yet incorrect approach would be to assume Y and $Y^\ominus$ are complementary, so they could bind together, forming a perfect double-strand. Since Y and $Y^\ominus$, i.e., $u3$-$u2$-$u1$ and $d1$-$d2$-$d3$ form the ends of the oligo X_i, if complementary, they would bind together prematurely, turning each X_i into a (defected) loop structure. To address that, instead of letting Y and $Y^\ominus$ to bind together directly, we let them 'cooperatively'' displace two upper strands O_1 and O_2 from the complex T as shown at the bottom of Figure 9.17. First, the strand Y reversibility displaces O_1 from the fuel gate L, producing the double-strand H, which cascades to the next displacement. Since the toehold $d1^*$ of the double-strand H is now open, it binds to $d1$ part of the strand $Y^\ominus$, which eventually displaces the

upper strand $d2$-$d3$ and together with Y form the waste $WT_{YY\ominus}$.

Overall, we transformed the input-weight integration part of the original CRN model of the linear chemical perceptron with 7 reactions to a deoxyribozyme-based circuit with 8 reactions: 6 cleavages (two per each input) and 2 DNA strand displacements. The original set of 8 species expanded to 16 species consisting of 6 deoxyribozymes and 10 oligos or single/double strands.

9.2.2 Discussion

We showed that deoxyribozymes are a natural choice for implementing catalytic reactions with DNA. As all other catalysts they drive reactions without being consumed, and hence promote reusability. By using deoxyribozymes in our designs we reduced the number of species and reactions significantly. Note that for certain parts, besides using deoxyribozyme-mediated catalysis, we incorporated also a few strand displacements, and so our deoxyribozyme implementations should be considered hybrid systems.

For the manual signalling delay line, instead of 15 displacement reactions and 49 strands needed for the DNA strand displacement implementation obtained by Soloveichik's transformation, the minimalistic deoxyribozyme version contains only 8 reactions and 19 species of which 6 are deoxyribozymes. The input-weight integration part of the linear chemical perceptron implemented by using deoxyribozymes shrank from 25 displacements and 70 strands to 8 reactions and 16 species of which 6 are deoxyribozymes.

9.3 SYSTEMS BIOLOGY AND CHEMICAL LEARNING

In an abstract chemical learning system, we can categorize species into four groups: input, output, functional, and feedback. The weight species of our chemical learners are specific examples of functional species, and the desired output and the penalty signal showcase

feedback species. The concentration of the functional species constitutes the behaviour of the system, and therefore it must be preserved during regular execution, so multiple inputs can be fed into the system sequentially. The most natural choice to achieve that is to make the functional species the catalysts that drive the input-to-output reactions, as we did. The learning module prescribes how the system should behave, and it adjusts the concentration of the functional species if needed.

Here we want to demonstrate that the application of the learning module goes beyond learning. In particular, an interesting dimension of chemical learning comes from the *Systems and Relational Biology* [86, 154]. In living organisms the metabolic transformation of substrates A to products B is catalysed by functional species (enzymes) f [122]. Rosen [131] distinguishes between the material causation $A \rightarrow B$ and the efficient causation $f \implies (A \rightarrow B)$ and asserts that life must be open to material, but closed to efficient causation. The problem here is that the metabolic catalysts undergo decay due to dilution and their finite stability, hence, an organism must continuously recreate them by an inner repair (replacement) mechanism, so the function species must be a product of metabolism as well. Rosen's (M, R) system (metabolic-repair or metabolic-replacement system) generalizes this idea and explicitly assumes the material link between the product of metabolism B and the catalysts f controlled by the replacement system Φ as shown in Figure 9.18(a). The replacement system is, however, physical, and therefore subject to decay. To avoid infinite regress, Rosen postulated the replication of Φ from the catalysts f with efficient causation of metabolites B obtaining the organization invariance of the system.

If we draw a diagram of a chemical learning system (Figure 9.18(b)) next to Rosen's diagram of life, we can see striking parallels. Note that to be consistent with the (M, R) formalism we renamed the species accordingly. The transformation of the input to the

output species in a chemical learning system is essentially a metabolism, although our interpretation of the output (product) has a *meaning* and is available for external measurement. Furthermore, the (re)creation of the functional species f in our model, called adaptation, is driven by the learning module Φ_L, as opposed to the term regulatory replacement operated by Φ. The main structural difference and also a simplification we adopted is that our functional species do not decay, so we could avoid explicit replacement. The learning module prescribes the expected behaviour, hence if the weights decayed and their concentrations diverted outside the desired region, and therefore produced incorrect output, the learning module would detect that and recover or stabilize the concentrations. Thus, if the learning module were invoked internally on a regular basis rather than an external cause, it would form its function factory, and consequently Rosen's replacement mechanism. This results in an interesting relation between individual learning, where the learning module is driven by an environment or a teacher, and population learning (evolution), where individuals share the hard-coded internal function factory with input-output pairs encoded in the genome. Although a function factory is in place and continually repairs an underlying function species, we still have to deal with the problem of its own degradation. Besides that, we can perhaps keep one instance of the learning module to maintain the function and one for individual learning (adaptation of function).

From a computer science perspective, the functional species guide the chemical execution and therefore represent the low-level (chemical) code. The specification of the learning module in terms of input and desired output pairs represents the high-level functional program. In the process of learning, a program compiles to the chemical code of the functional species concentrations, hence, the learning module introduces the compilation layer, bridging computer science and chemistry, allowing users without a background in chemistry to effectively program this chemical interface. Note that, as opposed to the or-
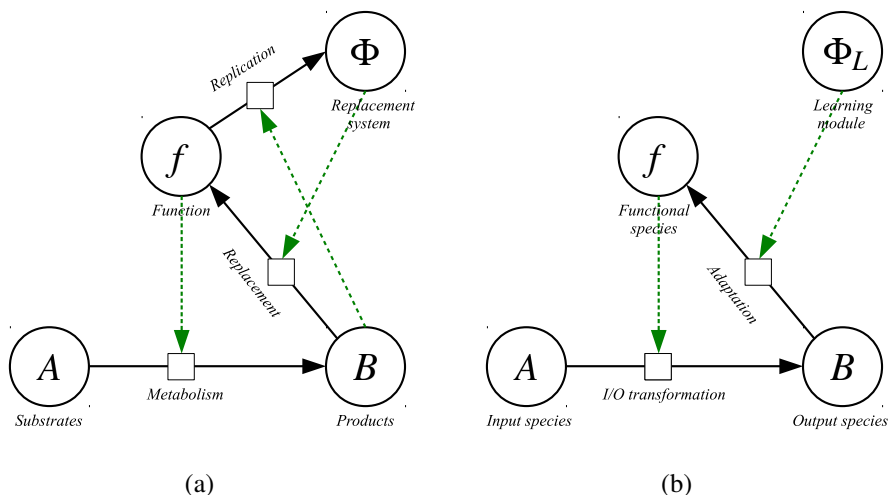
Figure 9.18: (a) Rosen's diagram showing organization invariance of life. The system is continually repaired and replicated as a product of its own metabolism. Solid arrows are material causations, whereas dashed arrows are efficient causations; (b) High-level diagram of an abstract chemical learning system. The concentration of functional species is adapted by the learning module implemented internally and operated externally by a trainer.

dinary procedural programming, the functional programming does not say *how* we want to achieve our goal in a step-by-step fashion, but specifies instead *what* behaviour we want our system to imitate.

Here, we illustrated that the learning is not just about adaptation to the feedback from the environment, but it encompasses the more general area of self-stabilization. Whether we should call this process learning, programming, regulation, replacement, or repair is strictly based on the purpose of what we consider a general function factory.

COEL—THE WEB-BASED CHEMISTRY SIMULATION FRAMEWORK

In this chapter we present a new enterprise chemistry simulation framework, COEL, which has been developed mainly as a part of this dissertation. COEL is the first web-based framework of its kind, and has been the sole simulation tool for modeling, evaluation and administration of all the chemical models mentioned in the dissertation.

The main motivation behind the development of the COEL framework is the often monotonous and inadequate management and execution of scientific models. Further, running simulations on multiple threads and CPUs requires non-trivial effort. Research avenues built on solid theoretical ideas often run into trouble because of a lack of appropriate tools and software, leading to unnecessary delays, implementation of proprietary (home-made) solutions for basic tasks and reinventions of standard design patterns. As is true with most desktop applications, most existing tools provide access to only a single user on a local machine, requiring version-management software to enable collaboration. General usability and visual appeal are usually low priorities. We argue that the way we work and conduct research must dramatically change to keep pace with the amount of data produced by simulations, to provide immediate and integrated visualization, and to enable geographically dispersed teams to work together on a single platform.

COllective cELlular computing (COEL) framework is the first web-based simulation framework for modeling and simulating chemical reaction networks (CRNs). COEL's

web client is immediately accessible without any installation or download. The computational load of simulations is handled by COEL's grid rather than the client's machine. Remote teams can share and manipulate chemical models in real time. Data is stored remotely and safely in COEL's database, which is backed up daily. In developing COEL we emphasized platform-wide visualization, providing quick and embedded insight for users.

It is important to emphasize the significance of COEL's database storage. Even though raw file storage (as opposed to structured databases) has been obsolete in industry for more than two decades, the scientific community still widely practices this approach. Storing data in files is not only ineffective, but its textual representation requires cumbersome parsing and tedious serialization for later structured searches, data mining or analysis. More so, files are inherently local, and without proper back-up, it is not uncommon that scientific data are lost. A recent study by Vines et al. in Current Biology [153] found that 80% of scientific data are lost within two decades, disappearing into old email addresses and obsolete storage devices. Alarmingly, the authors found that the average rate of data loss is 17% each year. Furthermore, because of private and local storing only 11% of the academic research in the literature was reproducible by the original research groups, as reported in Nature [26]. This is intuitively more prevalent in experimental science, but computer-based research is affected as well. We suggest that with current scientific approaches this problem will only worsen in the age of big data. We argue that storing all (even intermediate) models and results remotely and in a reliable long-term fashion, and making them accessible to the general scientific community should become the new standard. With remote data storage and a convenient web client, users do not have to deal with version-compatibility of data structures, as it is the case with traditional approaches. Since a new application release is deployed together with a central migration

of the database, version updates are worry-free for users.

Accessibility has two important consequences: collaboration and transparency. Using COEL, as with so-called 'cloud-based' web applications, individuals can work on different facets of the same project and see each other's modifications in real-time. This has allowed the authors of this paper, for example, to study the same system, run parameter evolutions and performance evaluations, modify simulation dynamics and so on from separate campuses. We have successfully applied COEL as a tool to model and evaluate various types of chemical perceptrons [20–22], chemical delay lines and time-series learners [19, 114], and random DNA circuits [18].

In this chapter we first discuss the state-of-the-art in chemistry simulation frameworks, then present COEL's functionality and technical architecture. We conclude with a discussion of COEL's place in the ecosystem of chemistry simulation frameworks, and the future of COEL. This work has been published in parts in [17].

10.1 RELATED WORK

COEL is not the first software made to simulate chemical reaction networks. There are already many programs which do so, and together the field of CRN simulators [36, 55, 70, 74, 134] offers a huge set of technical features, e.g., simulation options and statistical tools. Our goal with COEL was not (so much) to introduce new simulation algorithms or methods of analysis, but to include the most common and useful tools among CRN simulators in an intuitive and modern web-based package. This makes the tools of systems biology more accessible, and the research done with them more transparent, collaborative, and replicable.

COPASI [70] is arguably the most advanced and widely used tool. In a nutshell,

COPASI simulates a variety of chemical objects and allows for freedom in experiment design and statistical analysis. COPASI is quite feature rich, and could be considered the gold standard of CRN simulation frameworks. There are others worth mentioning, of course, such as those in the MATLAB Systems Biology Toolbox [134], and CellDesigner [55], which is a modeling tool for biochemical networks. Most of these tools share support for the SBML language for describing chemical systems [74], which as a standard has been a great boon to the field, enabling cross-platform migration.

Along with SBML support, most simulation environments share a core set of capabilities. Beyond basic deterministic ODE integration of CRNs (and stochastic reactions, a feature which COEL notably does not have), it is common to offer parameter optimization to help in the design of the networks themselves. Programs such as COPASI and CellDesigner can simulate a number of other biochemical objects of interest, such as cellular compartments. It is common to allow for various kinetic models of chemical interactions, such as Michaelis-Menten [108] and mass action [96].

In many kinds of frameworks, there is some tension between the depth of features and the features' accessibility, especially for highly technical applications such as CRN simulators. In addition to offering rich design capabilities, many developers of CRN simulators have the explicit motivation of reaching a large audience: The authors of COPASI said, "... the software needs to be available for the majority of scientists ..." (p. 3069, [70]). The authors of CellDesigner felt similarly, saying that they wish to "confer benefits to as many users as possible" (p. 1255, [55]). COEL automatically runs on any operating system with a web browser, including smartphones or tablets, so it is accessible anywhere in the world without any installation. Further, COEL's computational grid centrally runs any difficult tasks which might run slowly on clients' computers. We strongly believe that there is no more accessible paradigm for research tools than a web-based interface with

computation performed in the cloud.

10.2 FEATURES AND FUNCTIONALITY

COEL provides a unified web environment for the definition, manipulation, and simulation of chemical reaction networks. In this section, we will discuss COEL's functionality and application-wide features in detail.

10.2.1 Chemical Reaction Network Definition

At its most basic level, a chemical reaction network (CRN) consists of a finite set of chemicals and reactions. A CRN represents an unstructured macroscopic simulated chemistry, hence the species labeled with symbols are not assigned a molecular structure. The state of a CRN is represented by a vector of chemical species concentrations. Reaction rates define the strength or speed of reactions, as prescribed by kinetic laws–Michaelis-Menten [38] kinetics for catalytic reactions, and mass action kinetics [49] otherwise (Section 2.3.1).

COEL is consistent with these general CRN formalisms; next, we will describe details particular to COEL's implementation. COEL automatically computes appropriate rate functions once given numeric rate constants, yet it also allows users to define arbitrary rate functions using custom expressions over species labels, giving the user full freedom over the system's dynamics. Reactions can be uni- or bidirectional, and bidirectional reactions can have independent forward and backward rates.

Both species sets and reaction sets are extensible, in that new sets can be defined as expansions of old ones. This promotes reuse and modular design. Further, two CRNs can be merged combining their reactions and species into one network.

Species X X1C X1signal X2signal Y_aux B Sin Sout W0 W1 W1ne W2 W2ne Wne Wpo X1 X2 Y ✚

Reactions

✚ 🗑 ⊕

Do		Label	Group	Reaction	Forward Rate	Catalysts	Inhibitors
👁 ✑	☐	DL01		X → X1 + X1C	0.0225 * X1signal * X / (0.0020 + X)	X1signal	
👁 ✑	☐	DL02		X1C → X2	2.0000 * X2signal * X1C / (0.0706 + X1C)	X2signal	
👁 ✑	☐	DL03		X2signal → X1signal	1.3648 * X2signal		
👁 ✑	☐	DL04		X1signal →	0.0039 * X1signal		
👁 ✑	☐	R01	RG 1	Sin + Y →	0.4584 * Sin * Y		
👁 ✑	☐	R02	RG 2	Sin → Y	0.4459 * W0 * Sin / (1.8066 + Sin)	W0	
👁 ✑	☐	R03	RG 3	X1 + Y →	0.0203 * Y * X1		
👁 ✑	☐	R04	RG 4	X1 → Y	0.0378 * W1 * X1 / (2.5665 + X1)	W1	
👁 ✑	☐	R05	RG 3	X2 + Y →	0.0203 * Y * X2		

Figure 10.1: A partial description of a chemical reaction network in COEL. Species are listed at the top, and their reactions are presented in tabular form. The reactants and products are described in the third column, the forward reaction rates are in the fourth column, and any catalysts are in the fifth.

Figure 10.1 shows an example CRN in COEL, a memory-enabled chemical percep-tron [19]. The CRN's species, reactions, and reaction rates are presented in a unified view from which any of these objects can be easily edited in a few steps. Also, users can ex-port CRNs in Matlab, Octave, or SMBL formats if they wish to study their systems using different tools. It is also possible to import an SBML-defined CRN into COEL.

In imitation of biochemical cells or membranes, CRNs in COEL support hierarchical tree-like compartmentalization. Each compartment hosts an independent reaction set and vector of chemical concentrations. Compartments communicate with each other through permeation, formalized in what we call 'channels.' A channel works just like an ordinary reaction, except the reactant and product species reside in adjacent compartments. Among other things, this allows for modular design of chemical systems, where connected mod-ules reside in nested compartments, as shown in Figure 10.2.

Order	Id	Label	Channels	Permeability	Disassociate?
0	3574	Example subcompartment 1	• X1 ← X1' • Y → X1 • X2 ← X2'	• 0.3388 • 0.4387 • 0.3388	➡
1	3575	Example subcompartment 2	• X1 ← X1' • X2 ← X2' • Y → X2	• 0.3388 • 0.3388 • 0.4387	➡

Figure 10.2: COEL's representation of a permeation schema.

10.2.2 Chemical Reaction Network Simulation and Interaction Series

A major feature of COEL, in that it has been crucial to its early users and their work, is so-called interaction series. An interaction series allows the user to directly manipulate concentrations of species in the CRN. This feature is analogous to, though more capable than, automatic chemical injections into a reaction chamber. For compartment-extended CRNs, interaction series can be identically hierarchical, allowing for precise interaction with each component of the network.

Concentrations can be modified multiple times, not just initially. E.g., for iterative processes it is useful to define a set of periodic interactions. In specifying interactions, a user can define custom concentration-setting expressions, as well as custom variables for use in those expressions. For example, the bottommost interaction in Figure 10.3 injects species B (here a 'penalty species') at concentration 0.5 if the output species Y does not match AND of the original input concentrations, $X1\_inj$ and $X2\_inj$. The COEL Interaction Series API, as we call it, is then a scripted language that can describe a variety of complicated experimental scenarios without touching the underlying simulation-framework code. Thus end users have the freedom to manipulate the chemical system in a dynamic and safe way (basic expression validation is provided).

To actually simulate a CRN, a user runs a defined reaction network with a selected interaction series (which might be as simple as setting initial concentrations). Users can

choose from a number of non-adaptive and adaptive deterministic ODE solvers, such as Runge-Kutta4, Cash-Karp, and Fehlberg, to integrate their system. Upon running such a simulation, the user is by default shown an embedded chart of species concentrations over time (Figure 10.4). If further post-processing is required, full or filtered data could be easily exported into a CSV file.

Note that since ODE solvers are deterministic, two simulations using the same CRN and interaction series will always produce the same concentration traces if the interaction series is deterministic. That is, however, not the case for the interaction series in Figure 10.3, which uses random weight setting and randomly injects binary inputs at concentration 0 or 3. COEL does not currently have a feature to save random number seeds to exactly replicate simulations such as these.

Do	☐	Time	Variable/Species Assignment
👁 ✏	☐	0	• 3 → IN • W0 ← random(0.5,1.5) • W1 ← random(0.5,1.5) • W2 ← random(0.5,1.5)
👁 ✏	☐	1	• (rand()<0.5)*IN → X1_inj • (rand()<0.5)*IN → X2_inj • Sin ← IN • X1 ← X1_inj • X2 ← X2_inj • Y ← 0
👁 ✏	☐	51	• B ← (Y > 0.5 != (X1_inj != 0 || X2_inj != 0))

Figure 10.3: The details of a COEL interaction series. Left arrows denote the setting of species concentrations, and right arrows indicate assignments of user-defined variables. The interaction at time 100 does the following (note that at time 0 the variable *IN* is set to 3): first, the variables $X1\_inj$ and $X2\_inj$ are randomly set to 0 or 3 with equal probability. The concentration of Sin is set to 3, then the concentrations of $X1$ and $X2$ are set equal to their respective injection variables. Finally, Y is flushed from the system—its concentration is set to 0.
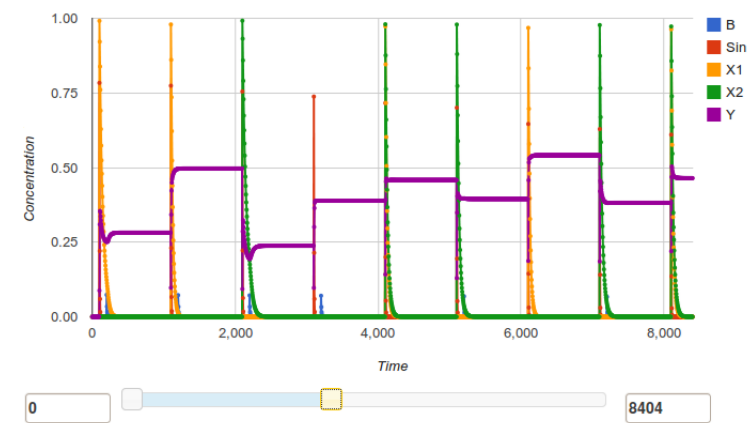
Figure 10.4: A chart showing concentration traces of 5 chemical species over time in COEL. In this case, an interaction series injects a random combination of $X1$ and $X2$ at concentration 1, every 1000 time steps.

10.2.3 Performance Evaluation and Dynamics Analysis

COEL provides a core set of tools for analyzing and modifying CRNs, enabling statistical record-keeping as well as the design of complex networks whose precise architecture is initially unknown to the user. COEL's basic interpretive tool is the "translation series," defined by the user in a similar manner to interaction series, described above. A single translation is a straightforward function of the current concentrations and any predefined constants, and can be Boolean or numeric in its output.

One can simply plot the output of a translation series to see the CRN's behavior through a certain lens, or use the series as the basis of evaluation and optimization. Because many CRNs involve a random component, especially in (but not limited to) their interaction series, COEL allows the user to run large batches of simulations and collect statistics based on these translation series.

Because it is usually difficult to precisely translate simulated chemistries into wet ones, COEL also offers perturbation analysis. Users can evaluate the performance of the CRN if a defined set of rates are randomly perturbed according to set parameters. This is

useful in measuring the robustness of a chemical system.

COEL also offers dynamics analyses with a detailed statistical view of an individual CRN simulation. This includes Lyapunov exponents, Derrida stability, time and spatial nonlinearity errors, and more; along with reports about the simulation itself, like how many species concentrations reached fixed points for given tolerance.

To allow maximum freedom in analysis, COEL offers CSV export of any raw data a user might produce. Every chart and data visualization in COEL is accompanied by a CSV export function, allowing the user to export either the data currently displayed on-screen (to replicate a chart or precisely modify its appearance) or the entire raw dataset, as shown in Figure 10.5.
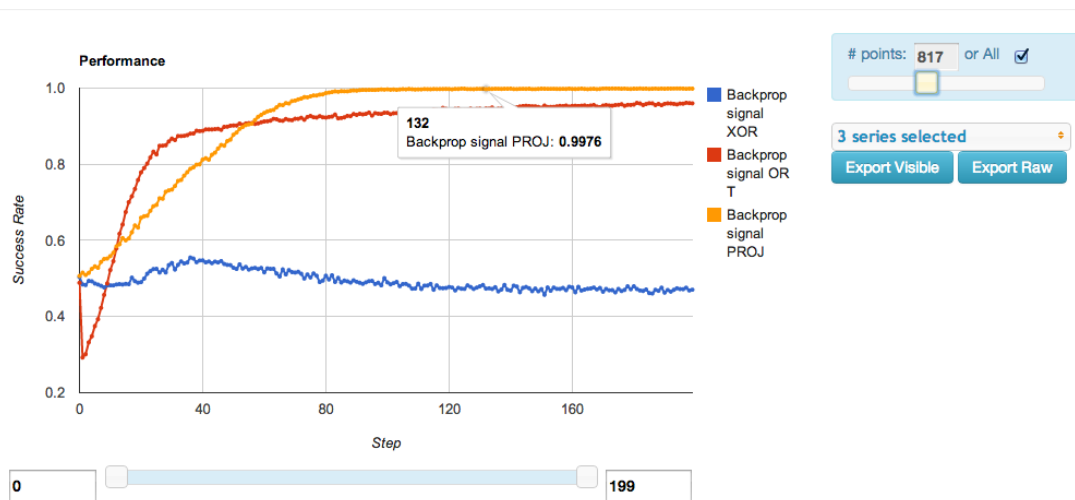
Figure 10.5: A chart of three separate performance evaluations, each one showing the performance of a binary chemical perceptron averaged over 10,000 repetitions for given interaction series representing desired binary function (XOR, OR, PROJ). Note the data export options on the right.

10.2.4 Rate Constant Optimization

With defined evaluation criteria, a user can optimize CRN's parameters with COEL's flexible genetic algorithm implementation. Users define the space to be optimized by se-
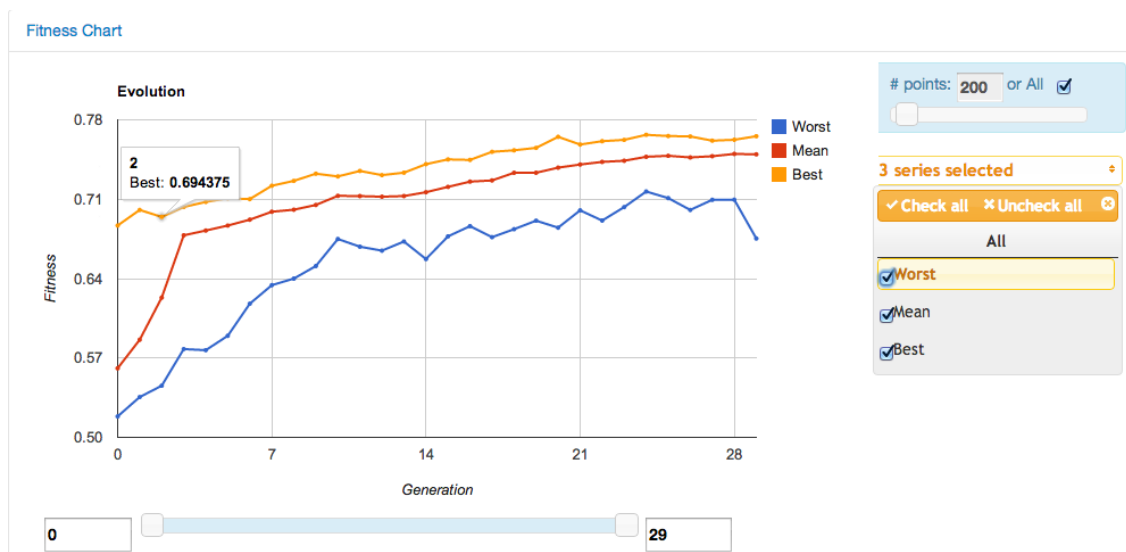
Figure 10.6: A chart of a population's fitness over time in a run of a particular GA. This plot displays several features shared by all plots in COEL, enabling modification of the plot without refreshing the web page: an x-axis slider to specify the plot's domain, a drop-down menu to select which series to display, and a slider to select the plot's resolution relative the data set.

lecting which reaction and channel permeation rates are to be modified, in what ranges, and under what constraints (e.g. several reaction rates can be fixed to each other). Chromosomes are then vectors of rate constants.

The parameters of COEL's GAs are easily modified, allowing for different rates of mutation, rules of reproduction, initial populations, and so on. Chromosomes can be selected to reproduce either deterministically with elite selection, or probabilistically relative the measured fitness of each chromosome. Reproduction can be sexual or asexual. In the former case, crossover between two chromosomes can be either one-point (i.e., in chromosomes of length n, the child's first $p \leq n$ genes are from one parent and the last $n - p$ are from the other), or a probabilistic shuffle. Supported mutation types are one-bit, two-bit, exchange and per-bit, with content replacement and perturbation options. COEL's GAs also support fitness renormalization, and selection of maximization or minimization of the target function (fitness vs. error).

10.2.5 DNA Strand Visualization and Displacement Reactions

COEL has a convenient web interface for visualizing DNA strands specified by the Microsoft Visual DSD syntax [90, 91], which decomposes single and (full or partial) double DNA strands into labeled subsequences called domains. Domains are classified as either long or short, also called toeholds. These DNA-strand images can be exported in the svg format, appropriate for publications and educational purposes alike. Note that the Microsoft Visual DSD web tool (unlike COEL) requires an installation of Microsoft Silverlight, whose support on Linux is problematic.

Furthermore, COEL can transform any CRN based on mass-action kinetics into a DNA strand-displacement circuit using the methods of Soloveichik et al. [140]. In strand displacement systems, populations of these species are typically represented by the populations of single-stranded DNA molecules. These interact with double-stranded gate complexes which mediate transformations between free signals. In a nutshell, the mass-action
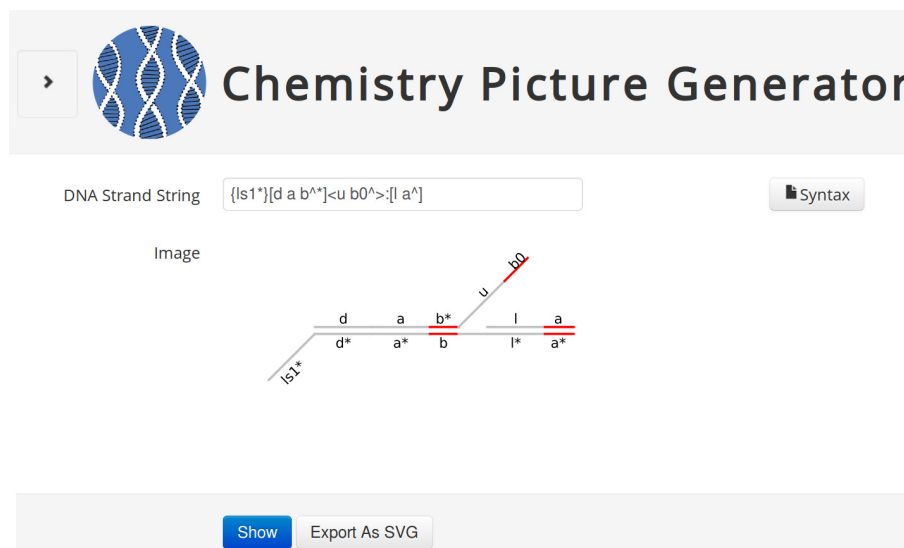
Figure 10.7: COEL's tool for visualizing DNA strands specified in Visual DSD. Red lines represent toeholds, and gray lines are long domains.

reaction $X_1 + X_2 \rightarrow X_3$ is translated to three displacement reactions $X_1 + L \leftrightharpoons H + B$ (a single strand X_1 displaces an upper strand B from the complex L), $X_2 + H \rightarrow O + W_1$ (a single strand X_2 displaces an upper strand O from the complex H), and finally $O + T \rightarrow X_3 + W_2$ (a single strand O displaces an upper strand X_3 from the complex T), where L, H, B, O, T, H are auxiliary fuel species, and W_1 and W_2 are waste products.

Once applied to a reaction set, the transformation produces a CRN with new intermediate species and reactions, describing displacements of single strands from partial or full double strands. Besides new reactions, COEL also specifies the DNA structure of each species in terms of numerically-labeled domains, the output of which is shown in Figure 10.8. This is a powerful tool for automatic translation of so-called *in silico* systems to feasible wet chemistries in a user-friendly way. The authors are not aware of any other CRN simulation framework that includes DNA strand displacement transformations as a part of their application toolbox.
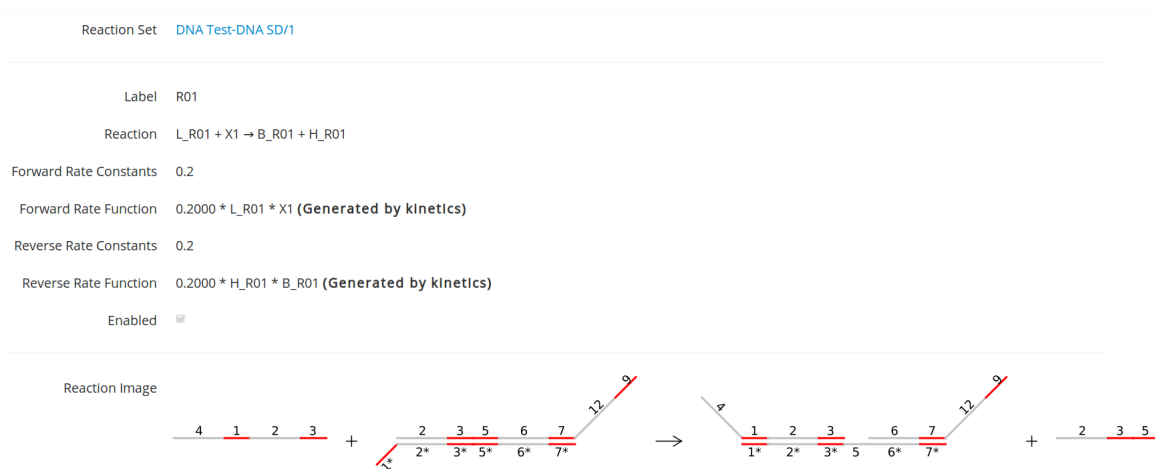
Figure 10.8: A DNA strand displacement reaction obtained by COEL's transformation of arbitrary CRNs into strand displacement circuits.

10.2.6   Random Chemical Reaction Network

COEL offers functionality to quickly make a random chemical reaction network with set specifications. User-defined parameters include the number of species, the number of reactions, the number of reactants and products in each reactions, and a random distribution of reaction constants; COEL meets all of these constraints with combinatorial design. For open systems the user can also specify influx and efflux constraints.

Furthermore, COEL also supports generation of random DNA-stand circuits [18] using single, full double, and partial double strands. Parameters for this function include number of single strands, ratio of upper to lower strands, ratio of upper strands with complements, (positive) normal distribution of partial double strands per upper strand, (positive) normal distribution of rate constants, ratio of influxes and effluxes, and distribution of rate constants. Based on a randomly generated ordering, DNA strands with higher order take precedence over lower-order strands in DNA-strand displacement reactions (Section 10.2.5). Also, note that the maximum number of strands that could bind together is two, which is justified by assuming that a single strand does not bind to partial double strand, but always displace its upper or lower part. We assume wet synthesis of these networks is possible by standard DNA sequence design [164].

10.2.7   Platform-wide Features

Numerous features of COEL are omnipresent throughout the platform, creating a familiar look-and-feel as well as providing intuitive access to common features. Throughout COEL, users input mathematical functions in the straightforward syntax of the Java Expression Parser (displayed in Figure 10.3), and those expressions are always validated by COEL before being input into any simulation. Views, such as COEL's list of reac-

tion sets or interaction series, have a common search and filter feature, allowing for easy navigation through huge sets of objects.

All charts in COEL are made with the Google Visualization API, and include sliders for domain selection and data filtering (see Figure 10.6), as well as CSV export options (see Figure 10.5). Finally, COEL has rudimentary user privacy protocols, where each user account is either a 'user' who can see only his/her own projects, or an 'admin' who can see every project on COEL. In order to share a project, a group of users currently have to have admin rights. We plan to expand privacy features in later versions.

10.3   ARCHITECTURE AND TECHNOLOGY

COEL's architecture is highly modular with strict separation of business logic and technological application aspects. Nowadays, the main challenge of enterprise application development is not programming per se but rather the integration of diverse technologies and libraries which each addresses different application needs. The absence of strict inter-modular / inter-layer dependencies enables quick and easy customization and replacement of technologies and providers.

At this level of abstraction only the domain objects, the data holders of business data, implemented as POJOs (Plain Java Objects), are shared among all application parts and layers. Figure 10.9 presents a high-level overview of COEL's architecture with call (request) pathways. On the very top we have two clients representing the only entry points to the application: the web client backed by Grails [5], jQuery [3] and Bootstrap [1] frameworks (discussed in Section 10.3.4), and the plain console client implemented in standard Java for "headless" scripting.

Based on user's requests, the clients call the services such as `ChemistryService`,

Figure 10.9: A high-level overview of COEL's architecture consisting of web and console clients, web servlet, services, business logic, persistence layer, and computational grid. The application (IoC) container holding the server-side of the application is implemented in Spring framework.

`Evolution Service`, and `UserManagementService` (Section 10.3.2) maintained by the Spring application container (Section 10.3.1), which then redirects either to a computational grid implemented on the top of GridGain HPC technology [2] (Section 10.3.3)

for distributed task execution, or to the persistence layer with DAOs (Data-Access Objects) and ORM (Object-Relation Mapping) provided by Hibernate [6] (Section 10.3.5). In addition, the web client controllers have a direct link to the persistence layer, which is beneficial especially for basic CRUD (Create, Read, Update, Delete) operations. At the very bottom a PostgreSQL [7] database stores and provides data on the demand of the persistence layer.

The business logic such as chemistry simulation and GA optimization is implemented mainly in the Scala language, leveraging both object-oriented and functional programming approaches. All technologies and libraries integrated into COEL are either opensource or free to use.

Table 10.1: A list of the acronyms used in this section.

Acronym	Description
JVM	Java Virtual Machine
ORM	Object-Relational Mapping
POJO	Plain Java Object
DAO	Data-Access Object
IoC	Inversion of Control
JEP	Java Expression Parser
JMS	Java Message Service
REST	Representational State Transfer
HPC	High Performance Computing
JDBC	Java Database Connectivity
SQL	Structured Query Language
PLSQL	Procedural Language/Structured Query Language
HQL	Hibernate Query Language

10.3.1 Application Container

The Spring Framework [8, 155] provides the COEL's core application infrastructure. Spring is a leading enterprise solution for Java maintained by the SpringSource community since 2002. Compared to Enterprise Java Beans, the Spring portfolio is less invasive

and more flexible. Spring is not an application server, it is just a set of libraries which can be used and deployed anywhere (like e.g., Tomcat and Jetty). It consists of several sub-projects which can be used separately or together as needed. Spring is a lightweight tool that shows how little is really needed for enterprise application development. It does not have strict dependencies, and it detaches technical and business concerns.

The IoC (Inversion of Control) container is a central part of the Spring Framework. It controls the creation, number of instances (with singleton and prototype scopes), life-cycle, inter-dependencies (loose-coupling or wiring) and general configuration of application components, modules, adapters, specific utility classes or in general any POJO whose creation and use should be maintained in the application context. Spring IoC is a simple and transparent glue or integrator of various components and frameworks which are provided either by Spring Portfolio itself or other parties.

The IoC container encourages the best practices of programming with interfaces, i.e., each bean (POJO object in the IoC container) should consist of an interface and implementation class. Therefore, each bean knows that it can talk to a different bean that does something specific, but not which type of object, how its functionality is implemented, nor how the call is carried out. The IoC container injects the dependencies into POJOs at the runtime, and so beans take care only about their business purpose, not creation (and maintenance) of their relationships.

This approach is superior to the factory design pattern because all dependencies get injected and configured through the application container (annotations and/or XML), however beans are not aware of the container's existence, i.e., unlike the factory pattern they do not need to call the application container in order to get their dependencies. The application code of Spring beans has little dependency on Spring itself. As a matter of fact, IoC is often described with the Hollywood principle: "Don't call us, we call you." Besides

Spring, other popular IoC containers include GUICE and Pico.

IoC abstraction results in modular, lightweight and layered architecture with loose-coupled pluggable components. Programmers are also encouraged to implement beans as thread-safe and stateless if possible, so several callers could safely query the same component without worrying about timing and/or call history.

Last but not least, Spring IoC enables COEL to become a truly test-driven project. Because of loose-coupling and dependency injections, our JUnit tests could switch to test (rather than production) application context and substitute for instance implementation classes that require remote access to production systems with mock objects.

10.3.2 Services

The service layer is the actual gateway to the business/functional part of the application. Services are callable functions provided to the clients (or outside world). COEL is divided into five functional modules, each exposed by a separate service interface (facade): `ChemistryService`, `EvolutionService`, `NetworkService`, `AnalysisService` and `UserManagementService`.

One of the most compelling reasons to use Spring for service management is its comprehensive transaction support. Spring provides a consistent abstraction for transaction management that integrates very well with various data access abstractions. For remote access, the service interfaces can be easily injected by appropriate stubs. Spring supports for example Remote Method Invocation (RMI), Spring's HTTP invoker, JAX-RPC, JAX-WS or JMS.

Since the web client runs as a part of the application context, i.e., it lives inside the same server-side JVM (Java Virtual Machine) as Spring, all service calls are local. On the other side, the console client runs as a separate process and its calls are remote. More

precisely, console clients requests are carried out by RESTFul Web Services and alternatively by JMS. In the future we might consider exposing a portion of services to 3rd parties, possibly other universities or teams, through REST.

10.3.3 Cloud Computing

COEL's computational grid has been built on top of the GridGain In-Memory Computing Platform [2]. The GridGain HPC (High Performance Computing) library implements a scalable low-latency zero-deployment computational grid, which fits seamlessly into our Spring-backed IoC container (Section 10.3.1).

COEL's grid currently consists of 30 nodes with around 750 cores. All nodes are hosted on Portland State University hardware, though the technology allows us to add any geographically remote resource, since the communication is carried out by TCP/IP protocol with optimized marshaling (serialization) of exchanged data. We plan to utilize existing grid technology to pool the resources with other geographically dispersed teams.

COEL's grid acts transparently, as a single computing resource. GridGain enables COEL's users to be more productive by eliminating the complexity of distributed computing. Regardless of a user's geographic location, they can add tasks to the grid from the COEL web page without much effort. When a user submits a task, after the chain of calls the request is ultimately received by the grid master node running within the application context. The task splits into many partial jobs, which are then distributed over the grid.

GridGain provides zero-deployment technology, so a new slave node (or a new task code) could be added to the grid on-the-fly by registering with the master node identified by the IP address or domain name. Therefore the grid's topology might change freely during its lifetime. COEL's grid supports several enterprise features contributing to effective and robust execution of jobs. The grid keeps track of various node statistics such as CPU

performance, execution time, and availability, which are constantly updated and utilized for adaptive job distribution such that high performing nodes obtain more jobs. Also, if a node disconnects from the grid, the exception is noted by a periodic heartbeat, and disconnected node's jobs are redistributed across the grid. Moreover, if a node finishes its execution sooner than expected and so it sits idle (its wait queue is empty), it steals jobs from other nodes.

Due to the communication and task initialization overhead we execute only nontrivial tasks on the grid, with compute times that can last seconds, hours, or days. The main grid tasks include chemical ODE simulations, dynamics analyses, and evolutionary optimizations of rate constants.

10.3.4   Web Client

COEL's web client is implemented in Grails [5], which is a powerful web 2.0 framework using the Groovy dynamic language for the Java Virtual Machine. JVM compatibility means that Java, Groovy, and Scala source compiles into Java byte code, hence these three languages are natively inter-callable. Grails follows the "Convention over Configuration" approach, which emphasizes standard (conventional) naming, binding and data flow, so the structure of the application is simply implied if it is not explicitly configured. This approach is heavily utilized in a function called scaffolding, which based on a domain object structure generates dynamically at runtime the controller with associated web pages, providing basic CRUD operations without any effort. As a matter of fact, we could build a COEL prototype web client just with a few lines of code. Grails internally uses Spring IoC for dependency injection and bean creation. Furthermore, Grails was officially incorporated into Spring portfolio at the end of 2008.

The web front-end relies heavily on Javascript provided by the jQuery library [3],

which makes UI interactive and intuitive and moves a part of data processing and visualization directly to the web browser. For instance, although COEL runs all simulations server-side, if a user wishes to see a chart, e.g., of species concentration traces, COEL sends the user raw data which is transformed into a chart by client-side Javascript using Google Charts API. For styling and some widgets we used the Bootstrap library [1] created by Twitter.
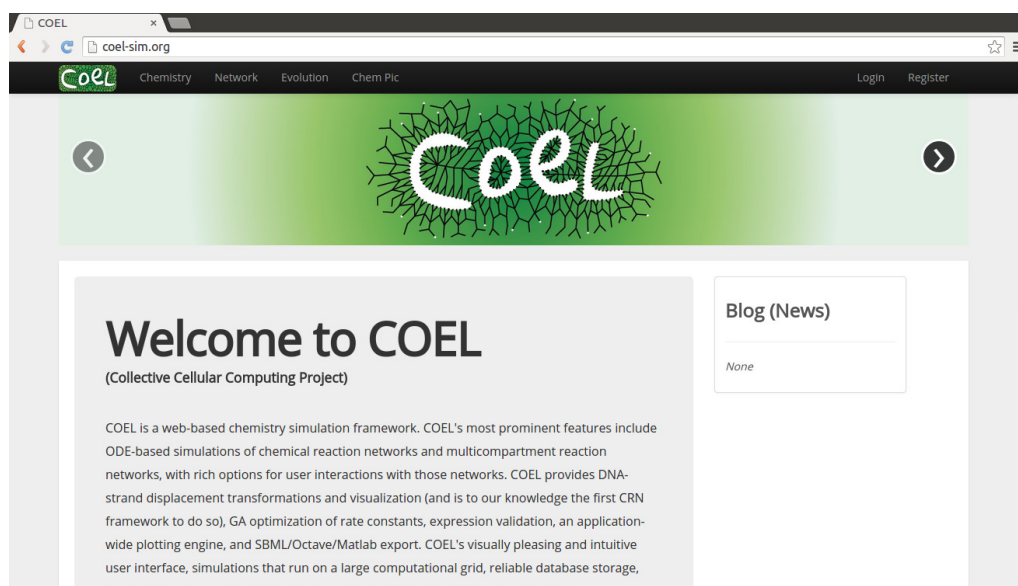
Figure 10.10: COEL's home (welcome) page. URL: coel-sim.org.

10.3.5 Persistence

The persistence layer consists of DAOs (Data-Access Objects) wrapping a storing, retrieving, deleting, and filtering functionality for domain objects. To map an object-oriented domain model to a traditional relational database we use Hibernate [6], an object-relational mapping (ORM) library for the Java language. DAOs and Hibernate are widely supported by Spring, which offers hooks for fast integration.

Hibernate solves Object-Relational impedance mismatch by replacing direct persistence-

related database accesses with high-level object handling functions. Hibernate provides declarative strategy for persisting data. Programmers define a mapping of columns, reference metadata and inheritance strategy mapping. Hibernate handles details about persistence implementation, like SQL statements and JDBC connection creation. To obtain data we use SQL or the Hibernate query language (HQL). The actual translation from the POJO to JDBC result set is automatic. Hibernate also uses various optimization strategies, such as cache and DB access optimization.

We believe that it is imperative to store data in a structured database, enabling prompt retrieval, searching and post-processing. PostgreSQL [7] is a mature open source database providing standard SQL/PLSQL language support with numerous additional features. The decision to select PostgreSQL as DB provider was driven mainly by the following factors: a lot of hands-on experience, a comprehensive console as well graphical UI (PgAdmin), an open source license, and support for array data types, useful for storing scientific vector data. The database model currently contains about 90 tables. To assure compatibility for each version of COEL we migrate data by a set of SQL scripts. Also, each day the whole database is dumped (backed-up), so we could restore the state of the DB to a certain date and time very quickly. That means our data is stored safely in structured and indexed format.

10.3.6 Build, Deploy, and Testing

To build COEL's project and to maintain its library dependencies, we use Apache Maven [4]. For a new application version we run a set of JUnit tests, which guarantee that the core functionality works as expected. After that, COEL is deployed to the Tomcat application server. Figure 10.11 shows a deployment schematic of COEL's components over several resources (machines), each running some part of the application: the database server, the

application server, and the cloud. Due to the extendability of the computational cloud, the number of resources is not bounded. Also, note that the database server and the application server are currently hosted on the same machine.

COEL currently has about 40 users, 5 of which are active, i.e., they access COEL on a daily basis. Once COEL will be available to the wider research community we expect the number of users to grow to hundreds, which would require more resources and more rigorous testing. If the users find a production issue or want to recommend a new feature, they will be able to submit a report through a Jira issue tracking system. More than 100 issues and new feature requests have be reported so far internally. Currently, the development of COEL is largely driven by the author's research needs.
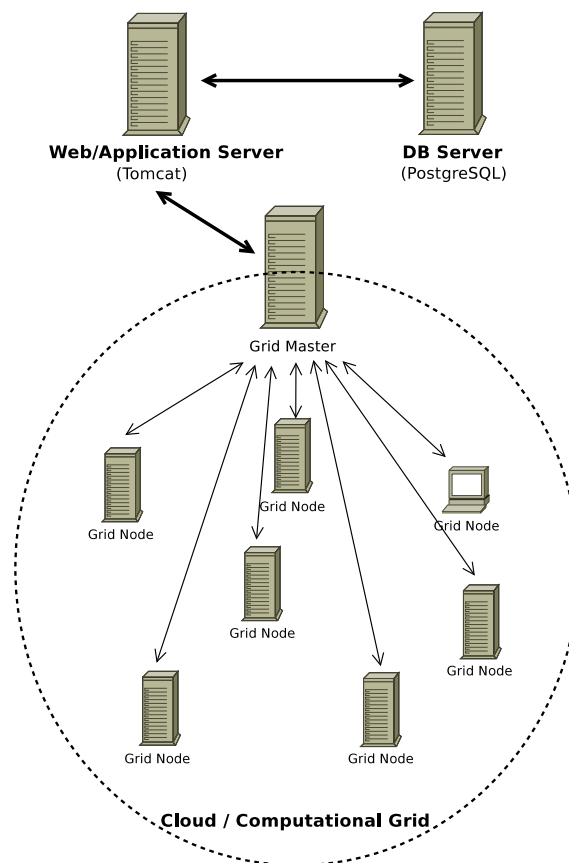
Figure 10.11: Diagram showing a physical deployment of COEL's components.

10.4 DISCUSSION

Here we presented a new web-based chemistry simulation framework, COEL. Its modern layered architecture includes a scalable computational grid, a user-friendly and interactive web UI, and the safe and transactive persistence of chemistry models and simulation results. Its wide range of features primarily target chemistry simulations, GA optimization of rate constants, performance evaluations, and dynamics analysis. We paid particular attention to general usability and lightweight and fluid layout, and embedded data visualization using Google's charting engine.

COEL can be used without any installation, and from any web browser. As such, it is easier to start using and has a larger potential audience than existing desktop-application based frameworks. Keeping COEL in the cloud allows for easy collaboration and sharing of results, and makes it simple to build upon another's work.

COEL's computational grid utilizes CPU resources only, however, it would be beneficial to extend the grid over GPUs as well. GridGain, our current computational grid library, does not provide native support for GPUs. On the other hand, we argue that reimplementing all tasks and business logic in (J)CUDA or OpenCL and maintaining two code branches would not be feasible. Therefore, we plan to explore transparent compilation mechanism such as Aparapi or RootBeer, where a single Java code compiles to CPU and GPU version transparently and gets executed based on resource availability.

Furthermore, we often face the situations when we want a newly submitted task to be executed as soon as possible, or we want to associate more CPU time to the tasks of a certain user. To achieve that we would like to assign priorities to the tasks based on their type and users' privileges.

As mentioned in Section 10.3.2 we might consider exposing certain services and rou-

tines through RestFul API so 3rd party applications could call, integrate and tailor COEL's functionality for their needs. Also, we plan to introduce more advanced sharing permissions, so each user could specify with which group or user he wants share the models and results for viewing and editing.

To improve the quality of chemistry ODE-based simulations we plan to integrate the standard LSODA solver. Also, to provide an alternative to the deterministic ODE solvers our goal is to introduce a stochastic simulator based on the Gillespie method [56]. The Gillespie method simulates each reaction step stochastically on a molecular level [79, 152]. It is computationally more demanding than ODE integration, however, it is physically more realistic, especially if the number of molecules in the system is low.

Last but not least, our vision for COEL is to become a common platform for diverse unconventional computing models. One step toward that goal is a new Network module, which will eventually simulate complex spatial, random, or layered networks with configurable node functions and interaction series.

11

CONCLUSION

In this dissertation, we extracted the essence of "learning and adaptation" formalized in the theory of neural networks and machine learning and transplanted it into the suit of chemical reaction networks (CRNs), macroscopic simulated chemistry driven by mass-action and Michaelis-Menten kinetics. Since chemical and neural network primitives are not compatible, we had to approach this reimplementation and mimicking problem from various angles.

As showed in Figure 11.1 and Table 11.1, we introduced several novel CRN models and constructions. Our models are the first CRNs capable of autonomous learning, i.e., supervised learning implemented internally and operated by a teacher. This work established a solid base for what we hope might become a new subfield interfacing chemistry and neural networks. The list of symbols and acronyms can be found on Page V.

We designed four binary chemical perceptrons: two symmetric, the WLP and the WRP, and two asymmetric, the SAPS and the TASP. These can learn 14 linearly separable binary functions perfectly. The asymmetric perceptrons are substantially smaller than the symmetric (around 50%) but are less robust to the perturbation of rate constants. Also, the asymmetric perceptrons learn by reinforcement (penalty signal) as opposed to desired output used by the symmetric perceptrons. The TASP, a thresholded version of the ASP, embeds an active thresholding, i.e., conditional amplification of the output by

incorporating Wilhelm's minimal bistable chemical system [159].

To store past input concentrations, we implemented two sequential delay lines with a linear structure and latency, the MDL and the BDL. A parallel-accessible delay line improves the previous models by providing a parallel access (a constant arbitrary small latency) and functions with no copy error.

To tackle more complicated (non-binary) scenarios, we modeled an analog chemical perceptron, the AASP, derived from the SASP. It can learn various linear and nonlinear functions of two inputs with an error (RNMSE) in the range $(0.103, 0.0.378)$. We demonstrated the modularity of the PDL by an integration with an AASP of two to five inputs, which tackled four time series.

We built a feedforward chemical neural network (FCNN), which consists of hierarchi-
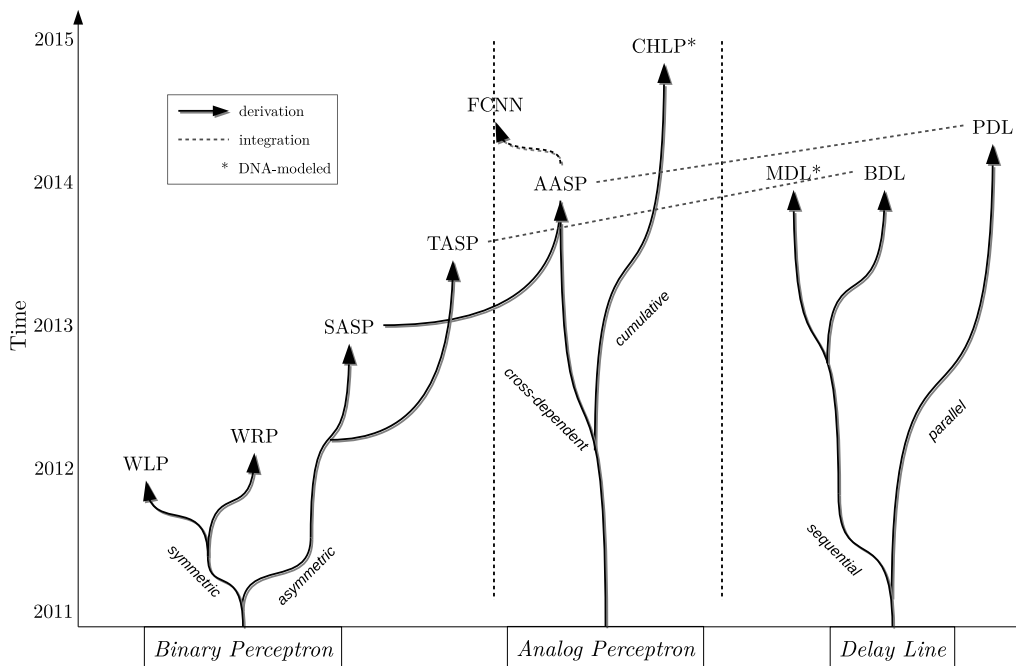
Figure 11.1: A high-level taxonomic tree of all our chemical models showing derivation paths, historical context, and integrations. Since the FCNN, which learns the binary functions, is not a perceptron but a multicompartment chemistry consisting of three analog perceptrons (AASPs), we placed it between the binary and analog perceptron models.

cal compartments communicating with each other through channel-mediated exchange of chemical species. By combining three modified AASPs into a two-to-one topology with the channels for the forward pass and the error backpropagation, we successfully addressed the non-linearly separable binary functions of XOR (100% accuracy) and XNOR (98.05% accuracy), which are beyond the capabilities of single chemical perceptrons.

To provide an insight into chemical learning, we rigorously analyzed the differential equations of the AASP and the CHLP and derived closed or approximative formulas for the input-weight integration and weight update. We showed that the formulas of the

Table 11.1: Overview of all our models, i.e., chemical binary and analog perceptrons (of two inputs), delay lines (of size three), and the feedforward chemical neural network (FCNN), showing the size (species / reactions), performance, and the most important features. The performance (error) of analog perceptrons is measured as RNMSE.

Model	Size (S / R)	Performance		Notes
Bin Perceptron		LS Bin Funs	Bin Funs	
WLP [21]	21/54	100%	94.71%	symmetric design, highly robust
				weights convert to output (cumbersome)
WRP [21]	14/30	100%	95.18%	symmetric design, highly robust
SASP [22]	12/16	99.5%	93.40%	asym. design, trained by penalty signal
TASP [22]	13/20	100%	94.80%	asym. design, trained by penalty signal
				active thresholding (bistability)
FCNN [27]	29/57	100%	99.88%	two hidden and one outer compartments
	16 channels			three modified AASPs, solves XOR
Ana Perceptron		Linear Fun	NARMA10	
AASP [20]	17/18	0.1034	0.7623	weight cross-dependence, nonscalable
				combined with PDL
CHLP	21/22	0.0013	0.3464	analytically designed, similar to NNLP
				additive, DNA-SD and deoxy modeled
Delay Line		Copy error	Latency	
MDL [114]	10/6	0	$O(n)$	sequential, manually operated
				DNA-SD and deoxy modeled
BDL [114]	10/6	exp	$O(n)$	sequential, signals backpropagated
PDL [19]	12/8	0	$O(1)$	parallel, only two signals, wait queues
				$O(n^2)$ species and reactions

CHLP match those of the formal neural network linear perceptron. The performance of the CHLP equals that of the neural network linear perceptron and exceeds the AASP by 94 or 437 times on average for the static functions depending on the error metrics. For our benchmark time series of linear weighted moving average, moving maximum, and highly nonlinear benchmark NARMA2 and NARMA10 tasks [15], the error is reduced by 8.37 (or 15.24) times to the $0.004 - 0.346$ RNMSE (or the $0.02 - 4.83\%$ SAMP). Even though we do not explicitly measure the generalization error, i.e., we do not distinguish between training and testing test, each training sample is drawn randomly from the full input-output domain (infinite for an analog setting). Our chemical models are therefore trained on previously unseen data and to succeed, they must approximate an underlying function.

An important product of this dissertation is the first cloud-based chemistry modeling tool, COEL, which stores all the models and ran all the required simulations. It is accessible for educational and research purposes at coel-sim.org.

To demonstrate that our models are implementable in wet chemistry, we applied two popular DNA-based techniques on one chemical learner, the linear chemical perceptron, and one chemical delay line, the manual signalling delay line. In particular, we employed DNA strand displacement circuitry [140, 166], which is generic but uses a large number of DNA fuel species, and deoxyribozyme gates [95, 145], which more naturally map to catalytic reactions and are reusable, but require ad-hoc adjustments to the original CRNs. Since the number of strands or deoxyribozymes in our two DNA-based implementations (molecular blue-prints) is within the complexity range of the state-of-the-art (experimentally constructed) circuits, we can assert that bringing our chemical designs into physical reality is plausible.

An important implication of our CRN-specified chemical learners is that any future

attempts to implement learning in wet chemistry would not need to reinvent the core reactions and the species roles, but rather focus on automatic or ad-hoc mapping to potentially arbitrary chemical substrate and reaction primitives. Since our CRN models focus on essential relations among species and kinetics and they abstract from actual molecular structure, they could accommodate different wet chemical substrates and implementation techniques.

11.1 APPLICATIONS

The importance of this research is the hope of a reprogrammable chemical computer. A transformation of CRNs to DNA displacement circuits, achieved mainly by the automatized techniques, such as Soloveichik's method [140] or Cardelli's two and three domain encodings [34, 35], is commonly associated with *programming*. Such methods posit that CRN, because of its abstract nature, is a programming language (symbolic and kinetic specification) and DNA circuit is its wet implementation (hardware). We argue that a use of the term programming in this context is misleading and overloaded. Designing a CRN for a given problem requires having new reactions and new species, DNA strands—chemical hardware—synthesized from scratch. In more constrained meaning of programming, a program is anything that specifies the behavior above the substrate or physical realization of the system. In chemical learning a single CRN with a single wet DNA implementation could be altered by the user's actions without touching or modifying CRN nor underlying DNA reaction mechanism.

Our chemical learners, like formal neural networks, operate in two modes: simply processing output when given an input, and learning via desired output or penalty signal and consequent backpropagation of errors. A wet implementation of a chemical learning

device, once trained to learn a task, could perform that task reliably as long as desired. The chemical computer could then, at any time, be retrained to perform any other task it supports.

Furthermore, the liquid nature of chemical computers and the possibility of constructing them from bio-compatible DNA strands open profound possibilities for patient-embedded biochemical computers. A fleet of cell-sized machines could monitor chemical concentrations in the bloodstream, and modulate the release of some drug accordingly.

With time-series integration, biochemical computers could keep a record of changing biological systems and detect and adapt to specific concentration patterns produced, e.g., by cancer cells in a host [162], acting as diagnostic aids and tools in preventative medicine and smart drug delivery [92]. Our approach could potentially replace hard-coded solutions and would allow reusing (retraining) chemical systems without redesigning them.

Learning and adaptation allow organisms to generalize and predict the ever-changing environment they live in, which leads to a competitive advantage for their survival and reproduction. Learning is therefore one of the pillars of life [25, 88]. As we discussed, the chemical learning's self-regulatory nature could be perceived as a programmable homeostasis. Understanding the organization principles of chemical learning might help us to track the origin of life [64], its prerequisites, and intermediates.

11.2 FUTURE WORK

Our work could be expanded in many ways. For instance, it would be intriguing to integrate standard chemical oscillators such as Lotka-Volterra [96, 97], the Brusellator [57], and the Oregonator [53] to drive the copy signals of a chemical delay allowing a system to sample and buffer input periodically. This inner clock (heart-beat) would enable ex-

tending chemical learners to sense continuous environment concentration. Further, from a systems biology perspective, an explicit decay of weight species, representing the state, would enforce the system to repair its functioning perhaps in combination with an oscillator triggering a learning procedure specified by an internalized target behavior. So far, we explored only the most basic multi-compartment topology with two subcompartments. A logical extension would be to introduce multi-nested compartments, hosting the linear chemical perceptrons with the input-weight integration and weight update modeled through the closed formulas, which would make the simulation time realistic.

Another future work is to employ so-called reservoir computing (RC) [33, 78, 98, 99, 136], a novel machine learning method based on recurrent neural networks. RC structurally consists of a fixed, randomly connected recurrent neural network, a reservoir, which acts as a set of high-dimensional filters with fading memory, and a memoryless readout layer, which is trained by supervised learning. The RC approach is relevant because for time-series prediction it is superior to classical recurrent networks and offers flexible implementation that could be expressed in various formalisms and substrates. The performance of a reservoir correlates to its dynamical properties, the most important of which is, the sensitivity to perturbations, rather than its structure. RC's loose structural assumptions therefore suggest that it could be expressed in a chemical form as well [59]. To implement a chemical reservoir we could randomly generate a chemical reaction network using graph-theoretical properties such as a species participation number, or generate a random DNA strand displacement system as we did in [18]. Learning, which occurs in a read-out layer, could be effectively carried out by a linear chemical perceptron or a feedforward chemical neural network.

A promise of this work and our long-term goal is to assemble a biomolecular learning machine in the laboratory. In collaboration with other teams we aim to sequence and

synthesize our DNA-specified models of the linear chemical perceptron and the manual signalling delay line and conduct a wet experiment as prescribed by our interaction series.

BIBLIOGRAPHY

[1] Bootstrap website. `http://getbootstrap.com`.

[2] Grid Gain website. `http://www.gridgain.com`.

[3] jQuery website. `http://jquery.com`.

[4] Official Apache Maven web site. `http://maven.apache.org`.

[5] Official Grails web site. `http://www.grails.org`.

[6] Official Hibernate web site. `http://hibernate.org`.

[7] Official PostgreSQL web site. `http://www.postgresql.org`.

[8] Official spring source web site. `http://www.springsource.org`.

[9] M. F. Abbod, D. A. Linkens, M. Mahfouf, and G. Dounias. Survey on the use of smart and adaptive engineering systems in medicine. *Artificial Intelligence in Medicine*, 26(3):179–209, 2002.

[10] L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, 1994.

[11] M. Amos. *Theoretical and experimental DNA computation (Natural Computing Series)*. Springer-Verlag, Secaucus, NJ, USA, 2003.

[12] D. Andre, F. H. Bennett III, and J. R. Koza. Evolution of intricate long-distance communication signals in cellular automata using genetic programming. In *Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems. Cambridge, MA: MIT Press*, pages 513–520, 1996.

[13] L. G. Arnaut, S. J. Formosinho, and H. Burrows. *Chemical kinetics: from molecular structure to chemical reactivity*. Elsevier, Amsterdam, 2007.

[14] D. Ashlock. *Evolutionary computation for modeling and optimization*, volume 103. Springer, 2006.

[15] A. F. Atiya and A. G. Parlos. New results on recurrent network training: unifying the algorithms and accelerating convergence. *Neural Networks, IEEE Transactions on*, 11(3):697–709, 2000.

[16] T. Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, 1996.

[17] P. Banda, D. Blount, and C. Teuscher. COEL: A web-based chemistry simulation framework. In S. Stepney and P. Andrews, editors, *CoSMoS 2014: Proceedings of the 7th Workshop on Complex Systems Modelling and Simulation*, pages 35–60. Luniver Press, 2014.

[18] P. Banda and C. Teuscher. Complex dynamics in random DNA strand circuits (extended abstract). In *The 19th International Conference on DNA Computing and Molecular Programming*. 2013.

[19] P. Banda and C. Teuscher. An analog chemical circuit with parallel-accessible delay line learning temporal tasks. In H. Sayama, J. Rieffel, S. Risi, R. Doursat,

and H. Lipson, editors, *ALIFE 14: Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems*, pages 482–489. MIT Press, 2014.

[20] P. Banda and C. Teuscher. Learning two-input linear and nonlinear analog functions with a simple chemical system. In O. H. Ibarra and L. Kari, editors, *Unconventional Computing and Natural Computing Conference*, volume 8553 of *Lecture Notes in Computer Science*, pages 14–26. Springer International Publishing, Switzerland, 2014.

[21] P. Banda, C. Teuscher, and M. R. Lakin. Online learning in a chemical perceptron. *Artificial life*, 19(2):195–219, 2013.

[22] P. Banda, C. Teuscher, and D. Stefanovic. Training an asymmetric signal perceptron through reinforcement in an artificial chemistry. *Journal of The Royal Society Interface*, 11(93):20131100, 2014.

[23] W. Banzhaf. The molecular traveling salesman. *Biological Cybernetics*, 64(1):7–14, Nov. 1990.

[24] S. Basu, Y. Gerchman, C. H. Collins, F. H. Arnold, and R. Weiss. A synthetic multicellular system for programmed pattern formation. *Nature*, 434(7037):1130–1134, 2005.

[25] M. Bedau. Artificial life: organization, adaptation and complexity from the bottom up. *Trends in Cognitive Sciences*, 7(11):505–512, Nov. 2003.

[26] C. G. Begley and L. M. Ellis. Drug development: Raise standards for preclinical cancer research. *Nature*, 483(7391):531–3, Mar. 2012.

[27] D. Blount, P. Banda, C. Teuscher, and D. Stefanovic. Feedforward chemical neural network: A compartmentalized chemical system that learns XOR. *IEEE Transactions on Neural Networks and Learning Systems*, 2014 (Under Review).

[28] R. S. Braich, N. Chelyapov, C. Johnson, P. W. K. Rothemund, and L. Adleman. Solution of a 20-variable 3-SAT problem on a DNA computer. *Science*, 296:499–502, 2002.

[29] D. Bray. Protein molecules as computational elements in living cells. *Nature*, 376(6538):307–312, July 1995.

[30] R. R. Breaker and G. F. Joyce. A dna enzyme that cleaves rna. *Chemistry & biology*, 1(4):223–229, 1994.

[31] R. Brooks. A robot that walks; emergent behaviors from a carefully evolved network. *Neural computation*, 1(2):253–262, 1989.

[32] N. E. Buchler, U. Gerland, and T. Hwa. On schemes of combinatorial transcription logic. *Proceedings of the National Academy of Sciences of the United States of America*, 100(9):5136–5141, Apr. 2003.

[33] L. Büsing, B. Schrauwen, and R. Legenstein. Connectivity, Dynamics, and Memory in Reservoir Computing with Binary and Analog Neurons. *Neural Computation*, 22(5):1272–1311, 2010.

[34] L. Cardelli. Strand algebras for DNA computing. In *DNA computing and molecular programming*, pages 12–24. Springer, Berlin Heidelberg, 2009.

[35] L. Cardelli. Two-domain DNA strand displacement. *Mathematical Structures in Computer Science*, 23(02):247–271, 2013.

[36] A. Castellini and V. Manca. Metaplab: a computational framework for metabolic P systems. In *Membrane Computing*, pages 157–168. Springer, Berlin Heidelberg, 2009.

[37] H.-J. K. Chiang, J.-H. R. Jiang, and F. Fagesy. Building reconfigurable circuitry in a biochemical world. In *Biomedical Circuits and Systems Conference (BioCAS)*, pages 560–563. IEEE, 2014.

[38] R. A. Copeland. *Enzymes: A practical introduction to structure, mechanism, and data analysis*. John Wiley & Sons, Inc., New York, New York, second edition, 2002.

[39] A. Cornish-Bowden and M. L. Cárdenas. Catalysis at the origin of life viewed in the light of the (m, r)-systems of Robert Rosen. *Proceedings of the Beilstein Bozen Symposium: Systems Chemistry*, pages 21–33, 2009.

[40] J. P. Crutchfield and P. Schuster. *Evolutionary dynamics: exploring the interplay of selection, accident, neutrality, and function*. Oxford University Press, 2003.

[41] A. P. de Silva, C. M. Dobbin, T. P. Vance, and B. Wannalerse. Multiply reconfigurable 'plug and play' molecular logic via self-assembly. *Chemical Communications*, (11):1386–1388, 2009.

[42] P. Dittrich. Chemical computing. In J.-P. Banâtre, P. Fradet, J.-L. Giavitto, and O. Michel, editors, *Unconventional Programming Paradigms*, volume 3566 of *Lecture Notes in Computer Science*, pages 21–32. Springer, Berlin / Heidelberg, 2005.

[43] P. Dittrich. Chemical computing. In J.-P. Banatre, P. Fradet, J.-L. Giavitto, and O. Michel, editors, *Unconventional Programming Paradigms*, pages 19–32. Springer, Berlin Heidelberg, 2005.

[44] P. Dittrich, J. Ziegler, and W. Banzhaf. Artificial chemistries - a review. *Artificial Life*, 7(3):225–275, 2001.

[45] H. El-Samad and M. Khammash. Modelling and analysis of gene regulatory network using feedback control theory. *International Journal of Systems Science*, 41(1):17–33, 2010.

[46] L. Elliott, D. Ingham, A. Kyne, N. Mera, M. Pourkashanian, and C. Wilson. Genetic algorithms for optimisation of chemical kinetics reaction mechanisms. *Progress in Energy and Combustion Science*, 30(3):297–328, Jan. 2004.

[47] I. R. Epstein and J. A. Pojman. *An introduction to nonlinear chemical dynamics: oscillations, waves, patterns, and chaos*. Oxford University Press, Oxford, 1998.

[48] P. Érdi and J. Tóth. *Mathematical models of chemical reactions: theory and applications of deterministic and stochastic models*. Manchester University Press, 1989.

[49] J. Espenson. *Chemical kinetics and reaction mechanisms*. McGraw-Hill, Singapore, 1995.

[50] D. Faulhammer. Molecular computation: RNA solutions to chess problems. *Proceedings of the National Academy of Sciences*, 97(4):1385–1389, Feb. 2000.

[51] H. Fellermann, S. Rasmussen, H.-J. Ziock, and R. V. Solé. Life cycle of a minimal protocell–a dissipative particle dynamics study. *Artificial Life*, 13(4):319–345, 2007.

[52] C. T. Fernando, A. M. L. Liekens, L. E. H. Bingle, C. Beck, T. Lenser, D. J. Stekel, and J. E. Rowe. Molecular circuits for associative learning in single-celled organisms. *Journal of the Royal Society, Interface / the Royal Society*, 6(34):463–9, May 2009.

[53] R. J. Field. Oscillations in chemical systems. IV. Limit cycle behavior in a model of a real chemical reaction. *The Journal of Chemical Physics*, 60(5):1877–1884, 1974.

[54] W. Fontana. Algorithmic chemistry. *Artificial life II*, pages 159–202, 1992.

[55] A. Funahashi, Y. Matsuoka, A. Jouraku, M. Morohashi, N. Kikuchi, and H. Kitano. CellDesigner 3.5: A Versatile Modeling Tool for Biochemical Networks. *Proceedings of the IEEE*, 96(8):1254–1265, Aug. 2008.

[56] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, Dec. 1977.

[57] P. Glansdorff and I. Prigogine. *Thermodynamic theory of structure, stability and fluctuations*, volume 53. Wiley, London, Mar. 1971.

[58] D. E. Goldberg et al. *Genetic algorithms in search, optimization, and machine learning*, volume 412. Addison-Wesley, Reading Menlo Park, 1989.

[59] A. Goudarzi, M. R. Lakin, and D. Stefanovic. DNA reservoir computing: A novel molecular computing approach. In D. Soloveichik and B. Yurke, editors, *DNA*

Computing and Molecular Programming, volume 8141 of *Lecture Notes in Computer Science*, pages 76–89. Springer International Publishing, Switzerland, 2013.

[60] M. A. Gutiérrez-Naranjo and M. J. Pérez-Jiménez. Membrane computing. chapter Hebbian Learning from Spiking Neural P Systems View, pages 217–230. Springer-Verlag, Berlin, Heidelberg, 2009.

[61] J. Halámek, V. Bocharova, S. Chinnapareddy, J. R. Windmiller, G. Strack, M.-C. Chuang, J. Zhou, P. Santhosh, G. V. Ramirez, M. A. Arugula, J. Wang, and E. Katz. Multi-enzyme logic network architectures for assessing injuries: digital processing of biomarkers. *Molecular BioSystems*, 6(12):2554–2560, Dec. 2010.

[62] S. Haykin. *Neural networks and learning machines*. Pearson, New Jersey, third edition, 2009.

[63] D. O. Hebb. *The organization of behavior*. John Wiley & Sons, New York, 1949.

[64] P. G. Higgs and N. Lehman. The rna world: molecular cooperation at the origins of life. *Nature Reviews Genetics*, 2014.

[65] A. Hjelmfelt and J. Ross. Chemical implementation and thermodynamics of collective neural networks. *Proceedings of the National Academy of Sciences of the United States of America*, 89(1):388–391, Jan. 1992.

[66] A. Hjelmfelt, E. D. Weinberger, and J. Ross. Chemical implementation of neural networks and Turing machines. *Proceedings of the National Academy of Sciences of the United States of America*, 88(24):10983–10987, Dec. 1991.

[67] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544, 1952.

[68] J. H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.

[69] M. H. Holmes. *Introduction to numerical methods in differential equations*. Springer-Verlag, New York, 2007.

[70] S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. COPASI–a COmplex PAthway SImulator. *Bioinformatics (Oxford, England)*, 22(24):3067–74, Dec. 2006.

[71] F. Horn and R. Jackson. General mass action kinetics. *Archive for rational mechanics and analysis*, 47(2):81–116, 1972.

[72] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[73] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[74] M. Hucka, a. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, a. P. Arkin, B. J. Bornstein, D. Bray, a. Cornish-Bowden, a. a. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J.-H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, a. Kremling, U. Kummer, N. Le Novere, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D.

Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, Mar. 2003.

[75] T. J. Hutton. Evolvable self-reproducing cells in a two-dimensional artificial chemistry. *Artificial Life*, 13(1), 2007.

[76] M. Ionescu, G. Păun, and T. Yokomori. Spiking neural P systems. *Fundamenta informaticae*, 71(2):1–28, 2006.

[77] M. Ionescu and D. Sburlan. Some applications of spiking neural P systems. *Computing and Informatics*, 27(3):515–528, 2008.

[78] H. Jaeger. 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, 2001.

[79] T. Jahnke and D. Altntan. Efficient simulation of discrete stochastic reaction systems with a splitting method. *BIT Numerical Mathematics*, 50(4):797–822, 2010.

[80] D. Jefferson, R. Collins, C. Cooper, M. Dyer, M. Flowers, R. Korf, C. Taylor, and A. Wanq. Evolution as a theme in artificial life: The genesys/tracker system. Technical report, Computer Science Department, University of California, 1990.

[81] H. Jiang, S. Salehi, M. Riedel, and K. Parhi. Discrete-Time Signal Processing with DNA. *ACS synthetic biology*, 2(5):245–254, May 2013.

[82] N. Jonoska. Biomolecular automata. In O. Shoseyov and I. Levy, editors, *NanoBioTechnology*, pages 267–299. Humana Press, Totowa, New Jersey, 2008.

[83] N. Kanopoulos and J. J. Hallenbeck. A first-in, first-out memory for signal processing applications. *Circuits and Systems, IEEE Transactions on*, 33(5):556–558, 1986.

[84] M. Kargol and A. Kargol. Mechanistic formalism for membrane transport generated by osmotic and mechanical pressure. *General Physiology and Biophysics*, 22(1):5168, 2003.

[85] J. Kim, J. J. Hopfield, and E. Winfree. Neural network computation by in vitro transcriptional circuits. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 17, pages 681–688. MIT Press, 2004.

[86] H. Kitano. Systems biology: a brief overview. *Science*, 295(5560):1662–1664, 2002.

[87] F. Kleinhans. Membrane permeability modeling: Kedem-Katchalsky vs a two-parameter formalism. *Cryobiology*, 37(4):271–289, 1998.

[88] D. E. Koshland. Special essay. The seven pillars of life. *Science (New York, N.Y.)*, 295(5563):2215–6, Mar. 2002.

[89] M. Lakin, A. Minnich, T. Lane, and D. Stefanovic. Towards a biomolecular learning machine. In J. Durand-Lose and N. Jonoska, editors, *Unconventional Computation and Natural Computation 2012*, volume 7445 of *Lecture Notes in Computer Science*, pages 152–163. Springer-Verlag, 2012.

[90] M. R. Lakin, S. Youssef, L. Cardelli, and A. Phillips. Abstractions for DNA circuit design. *Journal of the Royal Society, Interface / the Royal Society*, 9(68):470–86, Mar. 2012.

[91] M. R. Lakin, S. Youssef, F. Polo, S. Emmott, and A. Phillips. Visual DSD: a design and analysis tool for dna strand displacement systems. *Bioinformatics*, 27(22):3211–3213, 2011.

[92] D. A. LaVan, T. McGuire, and R. Langer. Small-scale systems for in vivo drug delivery. *Nature biotechnology*, 21(10):1184–91, Oct. 2003.

[93] M. Levy and A. D. Ellington. Exponential growth by cross-catalytic cleavage of deoxyribozymogens. *Proceedings of the National Academy of Sciences*, 100(11):6416–6421, 2003.

[94] C. Liébecq. *Biochemical nomenclature and related documents: A compendium.* Portland Press, London, United Kingdom, 2nd edition, 1992.

[95] J. Liu, Z. Cao, and Y. Lu. Functional nucleic acid sensors. *Chemical Reviews*, 109(5):1948–1998, 2009.

[96] A. J. Lotka. Undamped oscillations derived from the law of mass action. *Journal of the American chemical society*, 42(8):1595–1599, 1920.

[97] A. J. Lotka. Elements of physical biology. reprinted 1956 as elements of mathematical biology. 1924.

[98] M. Lukoševičius and H. Jaeger. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Computer Science Review*, 3(3):127–149, 2009.

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

[100] J. Macdonald, D. Stefanovic, and M. N. Stojanovic. Solution-phase molecular-scale computation with deoxyribozyme-based logic gates and fluorescent readouts. In *Fluorescent Energy Transfer Nucleic Acid Probes*, pages 343–363. Humana Press, Totowa, New Jersey, 2006.

[101] P. Maes. Modeling adaptive autonomous agents. *Artificial Life*, 1:135–162, 1994.

[102] S. Mailloux, J. Halámek, and E. Katz. A model system for targeted drug release triggered by biomolecular signals logically processed through enzyme logic networks. *The Analyst*, 139(5):982–986, Mar. 2014.

[103] S. Mailloux, O. Zavalov, N. Guz, E. Katz, and V. Bocharova. Enzymatic filter for improved separation of output signals in enzyme logic systems towards sense and treat medicine. *Biomaterials Science*, 2(2):184–191, 2014.

[104] H. Maturana and F. Varela. *Autopoiesis and cognition: The realization of the living*. D. Reidel Pub. Co., Dordrecht, Holland, 1980.

[105] H. T. Maune, S.-p. Han, R. D. Barish, M. Bockrath, W. A. Goddard III, P. W. Rothemund, and E. Winfree. Self-assembly of carbon nanotubes into two-dimensional geometries using dna origami templates. *Nature Nanotechnology*, 5(1):61–66, 2009.

[106] P. Mazzoni, R. A. Andersen, and M. I. Jordan. A more biologically plausible learning rule for neural networks. *Proceedings of the National Academy of Sciences*, 88(10):4433–4437, May 1991.

[107] T. Meyer and C. Tschudin. Flow management in packet networks through interacting queues and law-of-mass-action scheduling. Technical report, CS-2011-001, University of Basel, 2011.

[108] L. Michaelis and M. L. Menten. Die Kinetik der Invertinwirkung. *Biochem. Z.*, 49:333–369, 1913.

[109] A. P. Mills, B. Yurke, and P. M. Platzman. Article for analog vector algebra computation. *Biosystems*, 52(1-3):175–180, Oct. 1999.

[110] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1972.

[111] M. Minsky and P. Seymour. *Perceptrons*. MIT Press, Oxford, England, 1969.

[112] M. Mitchell. *An introduction to genetic algorithms*. MIT Press, Cambridge, 1996.

[113] M. Mitchell and S. Forrest. Genetic algorithms and artificial life. *Artificial Life*, 1(3):267–289, Jan. 1994.

[114] J. Moles, P. Banda, and C. Teuscher. Delay line as a chemical reaction network (in press). *Parallel Processing Letters*, 2014.

[115] J. S. Moles. Chemical reaction network control systems for agent-based foraging tasks. Master's thesis, Portland State University, 2015.

[116] G. Neat, H. Kaufman, and R. Roy. A hybrid adaptive control approach for drug delivery systems. In *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 1305–1306. IEEE, 1988.

[117] Nikaido, Hiroshi and Vaara, Marti. Molecular basis of bacterial outer membrane permeability. *Microbiological Reviews*, 49(1):1–32, 1985.

[118] N. Ono and H. Suzuki. Universal replication in a string-based artificial chemistry system. *Journal of Three Dimensional Images*, 16(4):154–159, 2002.

[119] Q. Ouyang. DNA solution of the maximal clique problem. *Science*, 278(5337):446–449, Oct. 1997.

[120] G. Paun, Y. Suzuki, H. Tanaka, and T. Yokomori. On the power of membrane division in P systems. *Theor. Comput. Sci.*, 324:61–85, Sep 2004.

[121] R. Pei, E. Matamoros, M. Liu, D. Stefanovic, and M. N. Stojanovic. Training a molecular automaton to play a game. *Nature Nanotechnology*, 5(11):773–777, Oct. 2010.

[122] G. Piedrafita, F. Montero, F. Morán, M. L. Cárdenas, and A. Cornish-Bowden. A simple self-maintaining metabolic system: robustness, autocatalysis, bistability. *PLoS computational biology*, 6(8):e1000872, Jan. 2010.

[123] W. Polifke, W. Geng, and K. Döbbeling. Optimization of rate coefficients for simplified reaction mechanisms with genetic algorithms. *Combustion and Flame*, 113(1-2):119–134, Apr. 1998.

[124] G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.

[125] G. Păun and G. Rozenberg. A guide to membrane computing. *Theoretical Computer Science*, 287(1):73–100, 2002.

[126] G. Păun, G. Rozenberg, and A. Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.

[127] L. Qian and E. Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332:1196–1201, 2011.

[128] L. Qian, E. Winfree, and J. Bruck. Neural network computation with DNA strand displacement cascades. *Nature*, 475(7356):368–372, July 2011.

[129] A. Rodan and P. Tino. Minimum complexity echo state network. *IEEE Transactions on Neural Networks*, 22(1):131–44, Jan. 2011.

[130] R. Rojas. *Neural networks: A systematic introduction*. Springer-Verlag, Berlin, 1996.

[131] R. Rosen. *Life itself: a comprehensive inquiry into the nature, origin, and fabrication of life*. Columbia University Press, 1991.

[132] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organisation in the brain. *Psychological Review*, 65:368–408, 1958.

[133] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct. 1986.

[134] H. Schmidt and M. Jirstrand. Systems Biology Toolbox for MATLAB: a computational platform for research in systems biology. *Bioinformatics (Oxford, England)*, 22(4):514–5, Feb. 2006.

[135] S. Schnell and C. Mendoza. Closed form solution for time-dependent enzyme kinetics. *Journal of Theoretical Biology*, 187:207–212, 1997.

[136] B. Schrauwen, D. Verstraeten, and J. V. Campenhout. An overview of reservoir computing: theory, applications and implementations. In *Proceedings of the 15th European Symposium on Artificial Neural Networks*, pages 471–482, Bruges, Belgium, 2007.

[137] O. Semenov, M. J. Olah, and D. Stefanovic. Mechanism of diffusive transport in molecular spider models. *Phys. Rev. E*, 83:021117, Feb 2011.

[138] S. W. Shin. *Compiling and verifying DNA-based chemical reaction network implementations*. PhD thesis, California Institute of Technology, 2011.

[139] B. Shirt-Ediss. Dynamical systems analysis of a protocell lipid compartment. In *Advances in Artificial Life. Darwin Meets von Neumann*, pages 230–239. Springer, 2011.

[140] D. Soloveichik, G. Seelig, and E. Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences of the United States of America*, 107(12):5393–5398, Mar. 2010.

[141] R. Stein. Some models of neuronal variability. *Biophysical journal*, 7(1):37–68, 1967.

[142] J. Stoer and R. Bulirsch. *Introduction to numerical analysis*. Springer-Verlag, New York, 2002.

[143] M. N. Stojanovic, P. de Prada, and D. W. Landry. Homogeneous assays based on deoxyribozyme catalysis. *Nucleic acids research*, 28(15):2915–8, Aug. 2000.

[144] M. N. Stojanovic, T. E. Mitchell, and D. Stefanovic. Deoxyribozyme-based logic gates. *Journal of the American Chemical Society*, 124(14):3555–3561, 2002.

[145] M. N. Stojanovic and D. Stefanovic. A deoxyribozyme-based molecular automaton. *Nature Biotechnology*, 21(9):1069–1074, Aug. 2003.

[146] M. N. Stojanovic and D. Stefanovic. Deoxyribozyme-based half-adder. *Journal of the American Chemical Society*, 125(22):6673–6676, 2003.

[147] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT Press, Cambridge, Massachusetts, 1998.

[148] K. Suzuki and T. Ikegami. Shapes and self-movement in protocell systems. *Artificial Life*, 15:59–70, 2009.

[149] Y. Suzuki and H. Tanaka. Symbolic chemical system based on abstract rewriting system and its halting property. In *Biocomputing and emergent computation: Proceedings of BCEC97*, pages 174–183. World Scientific Press, 1997.

[150] A. Tamsir, J. J. Tabor, and C. A. Voigt. Robust multicellular computing using genetically encoded NOR gates and chemical 'wires'. *Nature*, 469(7329):212–215, Dec. 2010.

[151] D. Tulpan, M. Andronescu, S. B. Chang, M. R. Shortreed, A. Condon, H. H. Hoos, and L. M. Smith. Thermodynamically based dna strand design. *Nucleic acids research*, 33(15):4951–4964, 2005.

[152] T. E. Turner, S. Schnell, and K. Burrage. Stochastic approaches for modelling in vivo reactions. *Computational biology and chemistry*, 28(3):165–178, July 2004.

[153] T. H. Vines, A. Y. K. Albert, R. L. Andrew, F. Débarre, D. G. Bock, M. T. Franklin, K. J. Gilbert, J.-S. Moore, S. Renaut, and D. J. Rennison. The availability of research data declines rapidly with article age. *Current biology*, 24(1):94–7, Jan. 2014.

[154] L. Von Bertalanffy. *General system theory: Foundations, development, applications*. George Braziller, New York, 1968.

[155] C. Walls. *Spring in Action*. Manning Publications, third edition edition, 2010.

[156] J. Wang and E. Katz. Digital biosensors with built-in logic for biomedical applications–biosensors based on a biocomputing concept. *Analytical and Bioanalytical Chemistry*, 398(4):1591–603, Oct. 2010.

[157] W. Wang, S. Li, L. Mair, S. Ahmed, T. J. Huang, and T. E. Mallouk. Acoustic propulsion of nanorod motors inside living cells. *Angewandte Chemie International Edition*, 53(12):3201–3204, Feb. 2014.

[158] J. D. Watson, F. H. Crick, et al. Molecular structure of nucleic acids. *Nature*, 171(4356):737–738, 1953.

[159] T. Wilhelm. The smallest chemical reaction system with bistability. *BMC systems biology*, 3(90), 2009.

[160] S. Wolfram. *Theory and Application of Cellular Automata*. Singapore: World Scientic, 1986.

[161] C. Wu, S. Wan, W. Hou, L. Zhang, J. Xu, C. Cui, Y. Wang, J. Hu, and W. Tan. A survey of advancements in nucleic acid-based logic gates and computing for appli-

cations in biotechnology and biomedicine. *Chemical Communications*, 51:3723–3734, 2015.

[162] Z. Xie, L. Wroblewska, L. Prochazka, R. Weiss, and Y. Benenson. Multi-input RNAi-based logic circuit for identification of specific cancer cells. *Science*, 333(6047):1307–1311, 2011.

[163] B. Yurke, A. J. Turberfield, A. P. Mills, F. C. Simmel, and J. L. Neumann. A dna-fuelled molecular machine made of dna. *Nature*, 406(6796):605–608, 2000.

[164] J. N. Zadeh, C. D. Steenberg, J. S. Bois, B. R. Wolfe, M. B. Pierce, A. R. Khan, R. M. Dirks, and N. A. Pierce. Nupack: analysis and design of nucleic acid systems. *Journal of computational chemistry*, 32(1):170–173, 2011.

[165] D. Y. Zhang. *Dynamic DNA strand displacement circuits*. PhD thesis, California Institute of Technology, 2010.

[166] D. Y. Zhang and G. Seelig. Dynamic DNA nanotechnology using strand-displacement reactions. *Nature chemistry*, 3(2):103–113, Feb. 2011.

[167] D. Y. Zhang and E. Winfree. Control of dna strand displacement kinetics using toehold exchange. *Journal of the American Chemical Society*, 131(47):17303–17314, 2009.

[168] M. Zhou, N. Zhou, F. Kuralay, J. R. Windmiller, S. Parkhomovsky, G. Valdés-Ramírez, E. Katz, and J. Wang. A self-powered "sense-act-treat" system that is based on a biofuel cell and controlled by Boolean logic. *Angewandte Chemie (International ed. in English)*, 51(11):2686–9, Mar. 2012.

APPENDICES

Appendix is divided into four sections, which contain extra materials about the chemical perceptrons (Appendix A), the feedforward chemical neural network that learns the logic functions including XOR and XNOR (Appendix B), the DNA strand displacement specifications of the chemical linear perceptron and the manual signalling delay line (Appendix C), and the various data that do not fit to the afformentioned categories (Appendix D). For easier reproducibility we provide all our models as ODEs in Octave/Matlab format, downloadable at `http://coel-sim.org/download`.

Appendix A

CHEMICAL PERCEPTRONS

Table A.1: The WLP's reactions with the best rate constants found by the GA, rounded to four decimal places. Reactions are divided into groups according to common functional characteristics.

#	Reaction	Catalysts	Inhibitors	Rates
1	$W_0^\oplus + E \rightarrow \overline{W}_0^\oplus + Y^1$ $W_0^\ominus + E \rightarrow \overline{W}_0^\ominus + Y^0$	$X_1^0, X_1^1, X_2^0, X_2^1$ $X_1^0, X_1^1, X_2^0, X_2^1$		0.0838, 3.7116, 0.2686, 0.4393
2	$W_1^\oplus + E \rightarrow \overline{W}_1^\oplus + Y^1$ $W_1^\ominus + E \rightarrow \overline{W}_1^\ominus + Y^0$ $W_2^\oplus + E \rightarrow \overline{W}_2^\oplus + Y^1$ $W_2^\ominus + E \rightarrow \overline{W}_2^\ominus + Y^0$	X_1^1 X_1^1 X_2^1 X_2^1		0.1630, 0.4358, 0.5058, 0.7404
3	$W_1^\oplus \rightarrow \overline{W}_1^\oplus$ $W_1^\ominus \rightarrow \overline{W}_1^\ominus$ $W_2^\oplus \rightarrow \overline{W}_2^\oplus$ $W_2^\ominus \rightarrow \overline{W}_2^\ominus$	X_1^0 X_1^0 X_2^0 X_2^0		0.0974, 4.5073
4	$\overline{W}_0^\oplus \rightarrow W_0^\oplus$ $\overline{W}_0^\ominus \rightarrow W_0^\ominus$ $\overline{W}_1^\oplus \rightarrow W_1^\oplus$ $\overline{W}_1^\ominus \rightarrow W_1^\ominus$ $\overline{W}_2^\oplus \rightarrow W_2^\oplus$ $\overline{W}_2^\ominus \rightarrow W_2^\ominus$		$X_1^0, X_1^1, X_2^0, X_2^1$ $X_1^0, X_1^1, X_2^0, X_2^1$ $X_1^0, X_1^1, X_2^0, X_2^1$ $X_1^0, X_1^1, X_2^0, X_2^1$ $X_1^0, X_1^1, X_2^0, X_2^1$ $X_1^0, X_1^1, X_2^0, X_2^1$	0.0093, 8.3625
5	$W_0^\oplus + W_0^\ominus \rightarrow \lambda$ $W_1^\oplus + W_1^\ominus \rightarrow \lambda$ $W_2^\oplus + W_2^\ominus \rightarrow \lambda$			0.2448
6	$Y^0 + Y^1 \rightarrow \lambda$			0.4249
7	$X_1^0 \rightarrow \lambda$ $X_1^1 \rightarrow \lambda$ $X_2^0 \rightarrow \lambda$ $X_2^1 \rightarrow \lambda$			0.0115
8	$Y^0 \rightarrow \lambda$ $Y^1 \rightarrow \lambda$			0.0009
9	$D^0 \rightarrow \lambda$ $D^1 \rightarrow \lambda$			0.0018
10	$D^0 \rightarrow W_0^\ominus$ $D^1 \rightarrow W_0^\oplus$	Y^1 Y^0		0.0710, 0.3033
11	$D^0 \rightarrow W_1^\ominus$ $D^0 \rightarrow W_2^\ominus$ $D^1 \rightarrow W_1^\oplus$ $D^1 \rightarrow W_2^\oplus$	Y^1, X_1^1 (AND) Y^1, X_2^1 (AND) Y^0, X_1^1 (AND) Y^0, X_2^1 (AND)		0.5000, 0.1955

Table A.2: The WRP's reactions with the best rate constants found by the GA, rounded to four decimal places. Reactions are divided into groups according to common functional characteristics.

#	Reaction	Catalysts	Rates
1	$X_1^0 \to Y^1$	$W_0^\oplus$	0.0972, 4.7912
	$X_1^0 \to Y^0$	$W_0^\ominus$	
	$X_2^0 \to Y^1$	$W_0^\oplus$	
	$X_2^0 \to Y^0$	$W_0^\ominus$	
	$X_1^1 \to Y^1$	$W_0^\oplus$	
	$X_1^1 \to Y^0$	$W_0^\ominus$	
	$X_2^1 \to Y^1$	$W_0^\oplus$	
	$X_2^1 \to Y^0$	$W_0^\ominus$	
2	$X_1^0 \to \lambda$	$W_0^\oplus$	0.0019, 5.0000
	$X_1^0 \to \lambda$	$W_0^\ominus$	
	$X_2^0 \to \lambda$	$W_0^\oplus$	
	$X_2^0 \to \lambda$	$W_0^\ominus$	
3	$X_1^1 \to Y^1$	$W_1^\oplus$	0.0081, 3.0102
	$X_1^1 \to Y^0$	$W_1^\ominus$	
	$X_2^1 \to Y^1$	$W_2^\oplus$	
	$X_2^1 \to Y^0$	$W_2^\ominus$	
4	$W_0^\oplus + W_0^\ominus \to \lambda$		0.5000
	$W_1^\oplus + W_1^\ominus \to \lambda$		
	$W_2^\oplus + W_2^\ominus \to \lambda$		
5	$Y^0 + Y^1 \to \lambda$		0.5000
6	$Y^0 \to \lambda$		0.0011
	$Y^1 \to \lambda$		
7	$D^0 \to \lambda$		0.0132
	$D^1 \to \lambda$		
8	$D^0 \to W_0^\ominus$	Y^1	0.0265, 1.8421
	$D^1 \to W_0^\oplus$	Y^0	
9	$D^0 \to W_1^\ominus$	Y^1, X_1^1 (AND)	0.3786, 0.0477
	$D^0 \to W_2^\ominus$	Y^1, X_2^1 (AND)	
	$D^1 \to W_1^\oplus$	Y^0, X_1^1 (AND)	
	$D^1 \to W_2^\oplus$	Y^0, X_2^1 (AND)	

Table A.3: The reactions with the best rate constants found by the GA, rounded to four decimal places for (a) the SASP MM, (b) the SASP MA, (c) the TASP MM, and (d) the TASP MA. Reactions are divided into groups according to common functional characteristics.

(a)

Group	Reaction	Catalyst	Rates
1	$S_{in} + Y \to \lambda$		0.1832
2	$S_{in} \to Y$	W_0	0.0637, 1.4774
3	$X_1 + Y \to \lambda$ $X_2 + Y \to \lambda$		0.0086
4	$X_1 \to Y$ $X_2 \to Y$	W_1 W_2	0.0112, 5.0
5	$P \to W^{\oplus}$		0.5402
6	$P \to W^{\ominus}$	Y	0.1465, 4.1054
7	$W^{\ominus} + W_0 \to \lambda$		0.0001
8	$W^{\oplus} \to W_0$		0.2135
9	$W^{\ominus} \to W_1^{\ominus}$ $W^{\ominus} \to W_2^{\ominus}$	X_1 X_2	0.1280, 0.0547
10	$W_1 + W_1^{\ominus} \to \lambda$ $W_2 + W_2^{\ominus} \to \lambda$		0.4459
11	$W^{\oplus} \to W_1$ $W^{\oplus} \to W_2$	X_1 X_2	0.4400, 0.8848

(b)

Group	Reaction	Rate
1	$S_{in} + Y \to \lambda$	0.4631
2	$S_{in} + W_0 \to Y + W_0$	0.1047
3	$X_1 + Y \to \lambda$ $X_2 + Y \to \lambda$	0.0287
4	$X_1 + W_1 \to Y + W_1$ $X_2 + W_2 \to Y + W_2$	0.0060
5	$P \to W^{\oplus}$	0.3795
6	$P + Y \to W^{\ominus} + Y$	0.0430
7	$W^{\ominus} + W_0 \to \lambda$	0.3593
8	$W^{\oplus} \to W_0$	0.0088
9	$W^{\ominus} + X_1 \to W_1^{\ominus} + X_1$ $W^{\ominus} + X_2 \to W_2^{\ominus} + X_2$	0.0348
10	$W_1 + W_1^{\ominus} \to \lambda$ $W_2 + W_2^{\ominus} \to \lambda$	0.1334
11	$W^{\oplus} + X_1 \to W_1 + X_1$ $W^{\oplus} + X_2 \to W_2 + X_2$	0.2032

(c)

Group	Reaction	Catalyst	Rates
1	$S_{in} + Y \to \lambda$		0.4584
2	$S_{in} \to Y$	W_0	0.4459, 1.8066
3	$X_1 + Y \to \lambda$ $X_2 + Y \to \lambda$		0.0203
4	$X_1 \to Y$ $X_2 \to Y$	W_1 W_2	0.0378, 2.5665
5	$P \to W^{\oplus}$		0.2082
6	$P \to W^{\ominus}$	Y	0.3137, 0.2370
7	$W^{\ominus} + W_0 \to \lambda$		0.018
8	$W^{\oplus} \to W_0$		0.1747
9	$W^{\ominus} \to W_1^{\ominus}$ $W^{\ominus} \to W_2^{\ominus}$	X_1 X_2	0.3036, 0.1282
10	$W_1 + W_1^{\ominus} \to \lambda$ $W_2 + W_2^{\ominus} \to \lambda$		0.2335
11	$W^{\oplus} \to W_1$ $W^{\oplus} \to W_2$	X_1 X_2	0.6000, 0.6235
12	$Y_{aux} \to 2Y$		1
13	$2Y \to Y + Y_{aux}$		1
14	$Y + Y_{aux} \to Y_{aux}$		0.5333
15	$Y \to \lambda$		0.3

(d)

Group	Reaction	Rate
1	$S_{in} + Y \to \lambda$	0.2922
2	$S_{in} + W_0 \to Y + W_0$	0.2731
3	$X_1 + Y \to \lambda$ $X_2 + Y \to \lambda$	0.0265
4	$X_1 + W_1 \to Y + W_1$ $X_2 + W_2 \to Y + W_2$	0.0088
5	$P \to W^{\oplus}$	0.0523
6	$P + Y \to W^{\ominus} + Y$	0.5019
7	$W^{\ominus} + W_0 \to \lambda$	0.0024
8	$W^{\oplus} \to W_0$	0.0037
9	$W^{\ominus} + X_1 \to W_1^{\ominus} + X_1$ $W^{\ominus} + X_2 \to W_2^{\ominus} + X_2$	0.6
10	$W_1 + W_1^{\ominus} \to \lambda$ $W_2 + W_2^{\ominus} \to \lambda$	0.518
11	$W^{\oplus} + X_1 \to W_1 + X_1$ $W^{\oplus} + X_2 \to W_2 + X_2$	0.4558
12	$Y_{aux} \to 2Y$	1
13	$2Y \to Y + Y_{aux}$	1
14	$Y + Y_{aux} \to Y_{aux}$	0.5333
15	$Y \to \lambda$	0.3

Appendix B

FEEDFORWARD CHEMICAL NEURAL NETWORK

Contained in these materials are the detailed numerical data, defining the FCNN as implemented in this paper. Table B.1 lists all chemical reactions and rates in each of our chemical neurons and the AASP. Table B.2 lists the cell-wall permeation channels between the neurons in the network, which enable feeding forward and backpropagation. Tables B.3 and B.4 list the ODEs that were integrated to simulate the FCNN, one for each species. Table B.5 lists the experimental protocol, i.e., the schedule of external injections into the FCNN that facilitated each learning iteration. Figure B.1 shows performance plots of several prototypes of the FCNN, as they attempt to learn the 16 binary functions. These prototypes used neurons derived from different single chemical perceptrons than the AASP, which was our final choice.

Table B.1: The reactions and rate constants of: (a) the original AASP, an analog asymmetric signal perceptron, (reproduced from [15]), (b) the hidden chemical neuron, and (c) the output chemical neuron. The hidden neuron is a modification of the AASP that omits the output–target-output comparison reactions and requires an extra reaction (highlighted in dark gray) for feeding forward the output. The output chemical neuron (also a modification of the AASP) uses the penalty species P and the threshold T for learning binary output, and propagates the error backwards using species $P_i^\oplus$ and $P_i^\ominus$. These species interact in modifications of the AASP's reactions (highlighted in light gray), as well as novel ones (dark gray). Note that the first 6 reactions in each set implement the input-weight integrations, the rest implement learning. The catalytic reactions have two rates: k_{cat} and K_m. All rate constants are rounded to four decimal places.

(a) AASP		(b) HCN		(c) OCN	
Reaction	Rates	Reaction	Rates	Reaction	Rates
$S_{in} + Y \to \lambda$	0.1800	$S_{in} + Y \to \lambda$	0.1800	$S_{in} + Y \to \lambda$	0.1800
$S_{in} \xrightarrow{W_0} Y + S_{in}^L$	0.5521, 2.5336	$S_{in} \xrightarrow{W_0} Y + S_{in}^L$	0.5521, 2.5336	$S_{in} \xrightarrow{W_0} Y + S_{in}^L$	0.5521, 2.5336
$X_1 + Y \to \lambda$ $X_2 + Y \to \lambda$	0.3905	$X_1 + Y \to \lambda$ $X_2 + Y \to \lambda$	0.3905	$X_1 + Y \to \lambda$ $X_2 + Y \to \lambda$	0.3905
$X_1 \xrightarrow{W_1} Y + X_1^L$ $X_2 \xrightarrow{W_2} Y + X_2^L$	0.4358, 0.1227	$X_1 \xrightarrow{W_1} Y + X_1^L$ $X_2 \xrightarrow{W_2} Y + X_2^L$	0.4358, 0.1227	$X_1 \xrightarrow{W_1} Y + X_1^L$ $X_2 \xrightarrow{W_2} Y + X_2^L$	0.4358, 0.1227
$\hat{Y} \to W^\oplus$	0.1884			$T \xrightarrow{S_L} E^\oplus$	0.1155, 1.9613
$Y \xrightarrow{S_L} W^\ominus$	0.1155, 1.9613			$Y \xrightarrow{S_L} E^\ominus$	0.1155, 1.9613
$Y + \hat{Y} \to \lambda$	1.0000			$Y + T \to \lambda$	5.0000
$W^\ominus \xrightarrow{S_{in}^L} W_0^\ominus$	0.600, 1.6697	$W^\ominus \xrightarrow{S_{in}^L} W_0^\ominus$	0.600, 1.6697	$W^\ominus \xrightarrow{S_{in}^L} W_0^\ominus$	0.600, 1.6697
$W_0 + W_0^\ominus \to \lambda$	0.2642	$W_0 + W_0^\ominus \to \lambda$	0.2642	$W_0 + W_0^\ominus \to \lambda$	0.2642
$W^\oplus \xrightarrow{S_{in}^L} W_0$	0.5023, 2.9078	$W^\oplus \xrightarrow{S_{in}^L} W_0$	0.5023, 2.9078	$W^\oplus \xrightarrow{S_{in}^L} W_0$	0.5023, 2.9078
$W^\ominus \xrightarrow{X_1^L} W_1^\ominus$ $W^\ominus \xrightarrow{X_2^L} W_2^\ominus$	0.1889, 1.6788	$W^\ominus \xrightarrow{X_1^L} W_1^\ominus$ $W^\ominus \xrightarrow{X_2^L} W_2^\ominus$	0.1889, 1.6788	$W^\ominus \xrightarrow{X_1^L} W_1^\ominus$ $W^\ominus \xrightarrow{X_2^L} W_2^\ominus$	0.1889, 1.6788
$W_1 + W_1^\ominus \to \lambda$ $W_2 + W_2^\ominus \to \lambda$	0.2416	$W_1 + W_1^\ominus \to \lambda$ $W_2 + W_2^\ominus \to \lambda$	0.2416	$W_1 + W_1^\ominus \to \lambda$ $W_2 + W_2^\ominus \to \lambda$	0.2416
$W^\oplus \xrightarrow{X_1^L} W_1$ $W^\oplus \xrightarrow{X_2^L} W_2$	0.2744, 5.0000	$W^\oplus \xrightarrow{X_1^L} W_1$ $W^\oplus \xrightarrow{X_2^L} W_2$	0.2744, 5.0000	$W^\oplus \xrightarrow{X_1^L} W_1$ $W^\oplus \xrightarrow{X_2^L} W_2$	0.2744, 5.0000
18 reactions, 20 rates		$Y \xrightarrow{S_F} F$	3.0000, 0.1000	$P \xrightarrow{E^\oplus} E^\oplus + W^\oplus$ $P \xrightarrow{E^\ominus} E^\ominus + W^\ominus$	1.0000, 1.0000
		16 reactions, 18 rates		$E^\oplus + E^\ominus \to \lambda$	5.0000
				$W^\oplus \xrightarrow{W_1} P_1^\oplus$ $W^\oplus \xrightarrow{W_2} P_2^\oplus$ $W^\ominus \xrightarrow{W_1} P_1^\ominus$ $W^\ominus \xrightarrow{W_2} P_2^\ominus$	0.3000, 0.5000
				25 reactions, 26 rates	

Table B.2: Permeation channels of the two-input, two-layer FCNN, where $^1\mathbf{C}$ and $^2\mathbf{C}$ are the two inner compartments. Groups 1 and 2 enable the inner perceptrons' input-weight integration, groups $3-5$ the outer perceptron's input-weight integration, and finally, groups 6 and 7 the inner perceptrons' learning (error backpropagation). All permeability constants are set to 1.

Group	Channels
1	$^1\mathbf{C} : (X_1 \leftarrow X_1')$
	$^2\mathbf{C} : (X_1 \leftarrow X_1')$
	$^1\mathbf{C} : (X_2 \leftarrow X_2')$
	$^2\mathbf{C} : (X_2 \leftarrow X_2')$
2	$^1\mathbf{C} : (S_{in} \leftarrow S_{in}')$
	$^2\mathbf{C} : (S_{in} \leftarrow S_{in}')$
3	$^1\mathbf{C} : (S_F \leftarrow S_F)$
	$^2\mathbf{C} : (S_F \leftarrow S_F)$
4	$^1\mathbf{C} : (F \rightarrow X_1)$
	$^2\mathbf{C} : (F \rightarrow X_2)$
5	$^1\mathbf{C} : (S_F \rightarrow S_{in})$
	$^2\mathbf{C} : (S_F \rightarrow S_{in})$
6	$^1\mathbf{C} : (W^{\oplus} \leftarrow P_1^{\oplus})$
	$^2\mathbf{C} : (W^{\oplus} \leftarrow P_2^{\oplus})$
7	$^1\mathbf{C} : (W^{\ominus} \leftarrow P_1^{\ominus})$
	$^2\mathbf{C} : (W^{\ominus} \leftarrow P_2^{\ominus})$
	Total 16

Table B.3: The full list of ODEs modelling the hidden output chemical neuron, whose reactions are shown in Table B.1(b). Note that these ODEs exclude the contributions or consumptions of the channels (Table B.2) mediating the communication between the FCNN's compartments.

$$\frac{d[S_{in}]}{dt} = -\frac{0.5521[W_0][S_{in}]}{2.5336 + [S_{in}]} - 0.1800[S_{in}][Y]$$

$$\frac{d[X_1]}{dt} = -\frac{0.4358[W_1][X_1]}{0.1227 + [X_1]} - 0.3905[X_1][Y]$$

$$\frac{d[X_2]}{dt} = -\frac{0.4358[W_2][X_2]}{0.1227 + [X_2]} - 0.3905[X_2][Y]$$

$$\frac{d[Y]}{dt} = \frac{0.5521[W_0][S_{in}]}{2.5336 + [S_{in}]} + \frac{0.4358[W_1][X_1]}{0.1227 + [X_1]} + \frac{0.4358[W_2][X_2]}{0.1227 + [X_2]}$$
$$- 0.1800[S_{in}][Y] - 0.3905[X_1][Y] - 0.3905[X_2][Y]$$
$$- \frac{0.1155[S_L][Y]}{1.9613 + [Y]} - 5.0[T][Y]$$

$$\frac{d[W_0]}{dt} = \frac{0.5023[S_{in}^L][W^\oplus]}{2.9078 + [W^\oplus]} - 0.2642[W_0^\ominus][W_0]$$

$$\frac{d[W_1]}{dt} = \frac{0.2744[X_1^L][W^\oplus]}{5.0 + [W^\oplus]} - 0.2416[W_1^\ominus][W_1]$$

$$\frac{d[W_2]}{dt} = \frac{0.2744[X_2^L][W^\oplus]}{5.0 + [W^\oplus]} - 0.2416[W_2^\ominus][W_2]$$

$$\frac{d[S_{in}^L]}{dt} = \frac{0.5521[W_0][S_{in}]}{2.5336 + [S_{in}]}$$

$$\frac{d[X_1^L]}{dt} = \frac{0.4358[W_1][X_1]}{0.1227 + [X_1]}$$

$$\frac{d[X_2^L]}{dt} = \frac{0.4358[W_2][X_2]}{0.1227 + [X_2]}$$

$$\frac{d[W^\oplus]}{dt} = -\frac{0.5023[S_{in}^L][W^\oplus]}{2.9078 + [W^\oplus]} - \frac{0.2744[X_1^L][W^\oplus]}{5.0 + [W^\oplus]} - \frac{0.2744[X_2^L][W^\oplus]}{5.0 + [W^\oplus]}$$
$$- \frac{0.3[W_1][W^\oplus]}{5.0 + [W^\oplus]} - \frac{0.3[W_2][W^\oplus]}{5.0 + [W^\oplus]}$$
$$+ \frac{1.0[E^\oplus][P]}{1.0 + [P]}$$

$$\frac{d[W^\ominus]}{dt} = -\frac{0.6[S_{in}^L][W^\ominus]}{1.6697 + [W^\ominus]} - \frac{0.1889[X_1^L][W^\ominus]}{1.6788 + [W^\ominus]} - \frac{0.1889[X_2^L][W^\ominus]}{1.6788 + [W^\ominus]}$$
$$- \frac{0.3[W_1][W^\ominus]}{5.0 + [W^\ominus]} - \frac{0.3[W_2][W^\ominus]}{5.0 + [W^\ominus]}$$
$$+ \frac{1.0[E^\ominus][P]}{1.0 + [P]}$$

$$\frac{d[W_0^\ominus]}{dt} = \frac{0.6[S_{in}^L][W^\ominus]}{1.6697 + [W^\ominus]} - 0.2642[W_0^\ominus][W_0]$$

$$\frac{d[W_1^\ominus]}{dt} = \frac{0.1889[X_1^L][W^\ominus]}{1.6788 + [W^\ominus]} - 0.2416[W_1^\ominus][W_1]$$

$$\frac{d[W_2^\ominus]}{dt} = \frac{0.1889[X_2^L][W^\ominus]}{1.6788 + [W^\ominus]} - 0.2416[W_2^\ominus][W_2]$$

$$\frac{d[T]}{dt} = -\frac{0.1155[S_L][T]}{1.9613 + [T]} - 5.0[T][Y]$$

$$\frac{d[P]}{dt} = -\frac{1.0[E^{\oplus}][P]}{1.0 + [P]} - \frac{1.0[E^{\ominus}][P]}{1.0 + [P]}$$

$$\frac{d[E^{\oplus}]}{dt} = \frac{0.1155[S_L][T]}{1.9613 + [T]} + \frac{1.0[E^{\oplus}][P]}{1.0 + [P]} - 5.0[E^{\ominus}][E^{\oplus}]$$

$$\frac{d[E^{\ominus}]}{dt} = \frac{0.1155[S_L][Y]}{1.9613 + [Y]} + \frac{1.0[E^{\ominus}][P]}{1.0 + [P]} - 5.0[E^{\ominus}][E^{\oplus}]$$

$$\frac{d[P1^{\oplus}]}{dt} = \frac{0.3[W_1][W^{\oplus}]}{5.0 + [W^{\oplus}]}$$

$$\frac{d[P2^{\oplus}]}{dt} = \frac{0.3[W_2][W^{\oplus}]}{5.0 + [W^{\oplus}]}$$

$$\frac{d[P1^{\ominus}]}{dt} = \frac{0.3[W_1][W^{\ominus}]}{5.0 + [W^{\ominus}]}$$

$$\frac{d[P2^{\ominus}]}{dt} = \frac{0.3[W_2][W^{\ominus}]}{5.0 + [W^{\ominus}]}$$

Table B.4: The full list of ODEs modelling the output chemical neuron, whose reactions are shown in Table B.1(c). Note that these ODEs exclude the contributions or consumptions of the channels (Table B.2) mediating the communication between the FCNN's compartments.

$$\frac{d[S_{in}]}{dt} = -\frac{0.5521[W_0][S_{in}]}{2.5336 + [S_{in}]} - 0.1800[Y][S_{in}]$$

$$\frac{d[X_1]}{dt} = -\frac{0.4358[W_1][X_1]}{0.1227 + [X_1]} - 0.3905[X_1][Y]$$

$$\frac{d[X_2]}{dt} = -\frac{0.4358[W_2][X_2]}{0.1227 + [X_2]} - 0.3905[X_2][Y]$$

$$\frac{d[Y]}{dt} = \frac{0.5521[W_0][S_{in}]}{2.5336 + [S_{in}]} + \frac{0.4358[W_1][X_1]}{0.1227 + [X_1]} + \frac{0.4358[W_2][X_2]}{0.1227 + [X_2]}$$
$$- 0.1800[S_{in}][Y] - 0.3905[X_1][Y] - 0.3905[X_2][Y]$$
$$- \frac{3.0[S_F][Y]}{0.1 + [Y]}$$

$$\frac{d[W_0]}{dt} = \frac{0.5023[S_{in}^L][W^\oplus]}{2.9078 + [W^\oplus]} - 0.2642[W_0^\ominus][W_0]$$

$$\frac{d[W_1]}{dt} = \frac{0.2744[X_1^L][W^\oplus]}{5.0 + [W^\oplus]} - 0.2416[W_1^\ominus][W_1]$$

$$\frac{d[W_2]}{dt} = \frac{0.2744[X_2^L][W^\oplus]}{5.0 + [W^\oplus]} - 0.2416[W_2^\ominus][W_2]$$

$$\frac{d[S_{in}^L]}{dt} = \frac{0.5521[W_0][S_{in}]}{2.5336 + [S_{in}]}$$

$$\frac{d[X_1^L]}{dt} = \frac{0.4358[W_1][X_1]}{0.1227 + [X_1]}$$

$$\frac{d[X_2^L]}{dt} = \frac{0.4358[W_2][X_2]}{0.1227 + [X_2]}$$

$$\frac{d[W^\oplus]}{dt} = -\frac{0.5023[S_{in}^L][W^\oplus]}{2.9078 + [W^\oplus]} - \frac{0.2744[X_1^L][W^\oplus]}{5.0 + [W^\oplus]} - \frac{0.2744[X_2^L][W^\oplus]}{5.0 + [W^\oplus]}$$

$$\frac{d[W^\ominus]}{dt} = -\frac{0.6[S_{in}^L][W^\ominus]}{1.6697 + [W^\ominus]} - \frac{0.1889[X_1^L][W^\ominus]}{1.6788 + [W^\ominus]} - \frac{0.1889[X_2^L][W^\ominus]}{1.6788 + [W^\ominus]}$$

$$\frac{d[W_0^\ominus]}{dt} = \frac{0.6[S_{in}^L][W^\ominus]}{1.6697 + [W^\ominus]} - 0.2642[W_0][W_0^\ominus]$$

$$\frac{d[W_1^\ominus]}{dt} = \frac{0.1889[X_1^L][W^\ominus]}{1.6788 + [W^\ominus]} - 0.2416[W_1^\ominus][W_1]$$

$$\frac{d[W_2^\ominus]}{dt} = \frac{0.1889[X_2^L][W^\ominus]}{1.6788 + [W^\ominus]} - 0.2416[W_2][W_2^\ominus]$$

$$\frac{d[F]}{dt} = \frac{3.0[S_F][Y]}{0.1 + [Y]}$$

Table B.5: The interaction series that represents an experimental protocol of FCNN learning for the target binary function $f(x_1, x_2)$. The random weight setting at time 0 is performed only initially, the rest of injections and assignments defined at time 100, 140, and 200 are repeated with periodicity 500 each learning iteration. The learning parameters are defined as follows: penalty signal concentration (learning rate) $\alpha = 1$, annealing rate $k = 0.0008$, and threshold concentration $\theta = 0.6$.

Time	Injections/Assignments
0	pick $[W_0] \in (0.5, 1.5)$
	pick $[W_1] \in (0.5, 1.5)$
	pick $[W_2] \in (0.5, 1.5)$
100	pick $x_1 \in \{0, 1\}$
	pick $x_2 \in \{0, 1\}$
	$[S'_{in}] = 0.5$
	$[X'_1] = x_1$
	$[X'_2] = x_2$
	$[S^L_{in}] = [X^L_1] = [X^L_2] = 0$
	$[E^\oplus] = [E^\ominus] = 0$
	$[S_L] = 0$
140	$[S'_F] = 1$
200	$incorrect = [Y] > \theta \neq f(x_1, x_2)$
	$\alpha = (1 - k)\alpha$
	$[T] = \theta \quad if\ incorrect$
	$[P] = \alpha \quad if\ incorrect$
	$[S_L] = 1 \quad if\ incorrect$
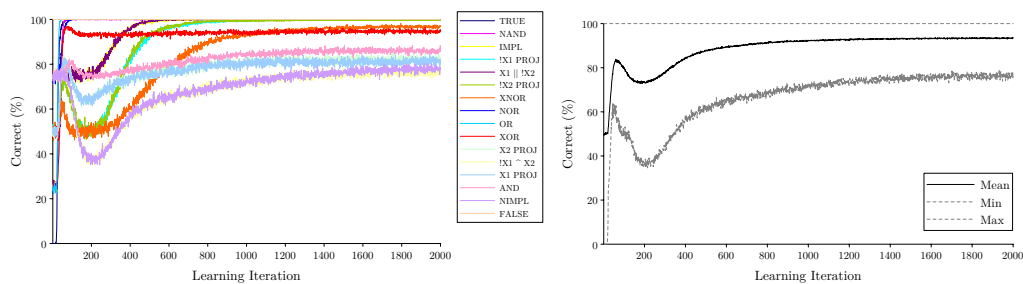
(a) OCN: SASP, HCN: AASP. ($P = 1.0$, $k = 0.995$). Mean final accuracy: 93.5%.
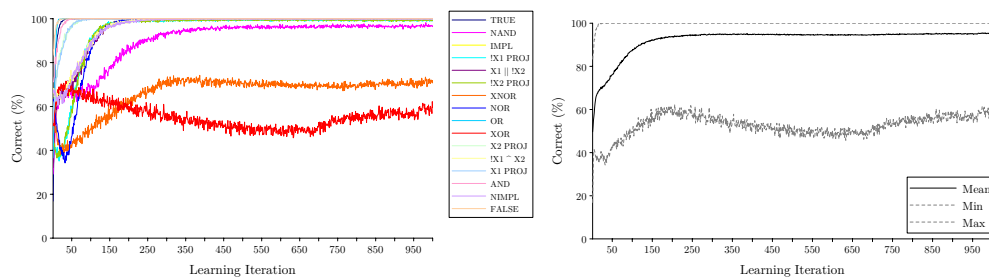
(b) OCN: TASP, HCN: AASP. ($P = 1.0$, $k = 0.99$). Mean final accuracy: 95.4%.

Figure B.1: Shown are accuracy/learning iteration plots for two prototypes of the FCNN. Under each plot, we describe the model that generated it in terms of: the perceptron module used as the output chemical neuron (OCN), that used as the hidden chemical neuron (HCN), the initial penalty concentration P, and the annealing rate k.
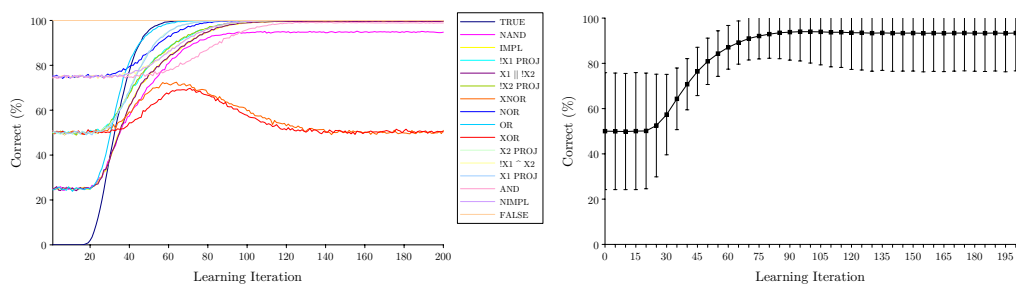
(a) SASP. Mean final accuracy: 93.4%.
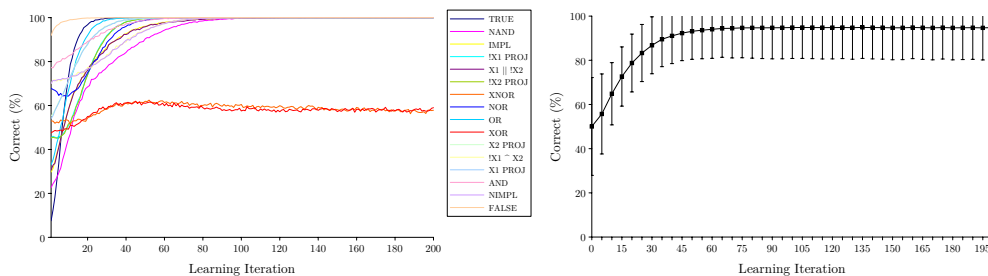
(b) TASP. Mean final accuracy: 94.8%.

Figure B.2: Shown are accuracy/learning iteration plots for two single binary chemical perceptrons, the SASP and the TASP.

DNA STRAND DISPLACEMENT

$$S_{in} + G_{R01} \rightarrow WT1_{R01} + O_{R01}$$

$$O_{R01} + T_{R01} \rightarrow WT2_{R01} + Y^{\ominus} + S_{in}^{L}$$

$$S_{in} + L_{R02} \leftrightarrow H_{R02} + B_{R02}$$

$$W_0 + H_{R02} \rightarrow WT1_{R02} + O_{R02}$$

$$O_{R02} + T_{R02} \rightarrow WT2_{R02} + Y + S_{in}^{L} + W_0$$
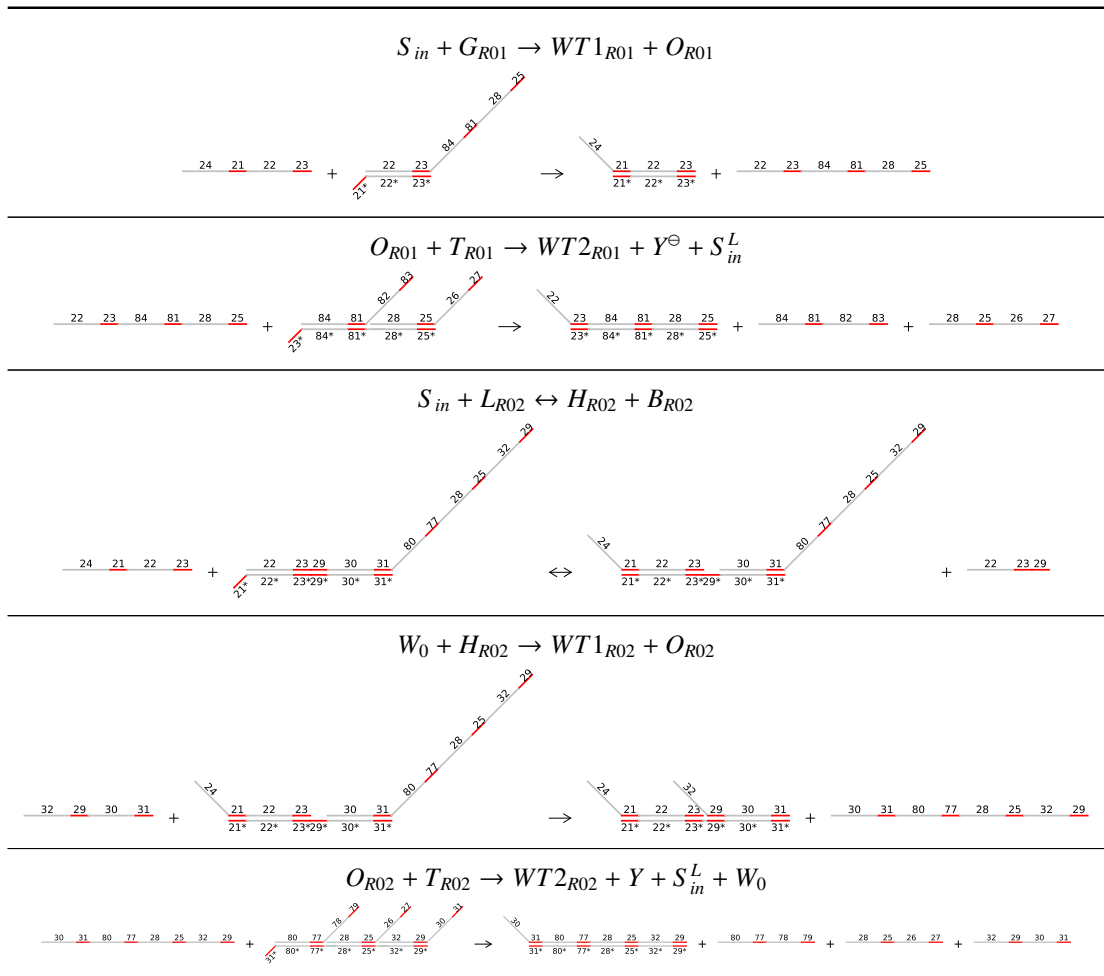
$$X_1 + G_{R03} \rightarrow WT1_{R03} + O_{R03}$$
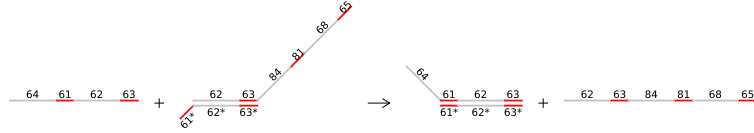
$$O_{R03} + T_{R03} \rightarrow WT2_{R03} + Y^{\ominus} + X_1^L$$
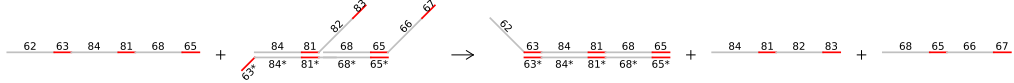
$$X_1 + L_{R04} \leftrightarrow H_{R04} + B_{R04}$$
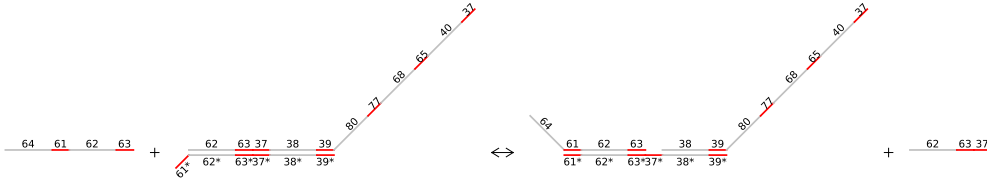
$$W_1 + H_{R04} \rightarrow WT1_{R04} + O_{R04}$$
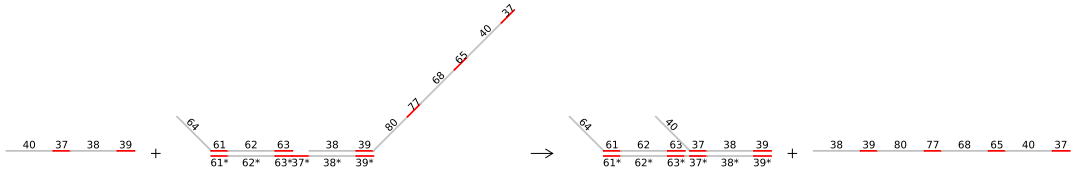
$$O_{R04} + T_{R04} \rightarrow WT2_{R04} + Y + X_1^L + W_1$$
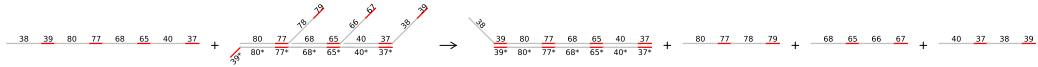
$$X_2 + G_{R05} \rightarrow WT1_{R05} + O_{R05}$$

$$O_{R05} + T_{R05} \rightarrow WT2_{R05} + Y^{\ominus} + X_2^L$$

283
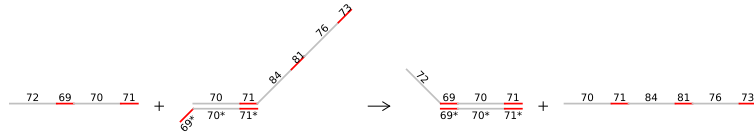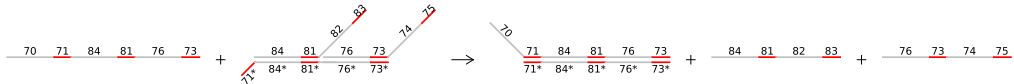
$$X_2 + L_{R06} \leftrightarrow H_{R06} + B_{R06}$$
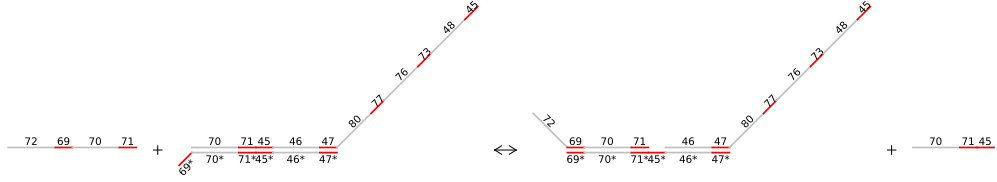
$$W_2 + H_{R06} \rightarrow WT1_{R06} + O_{R06}$$
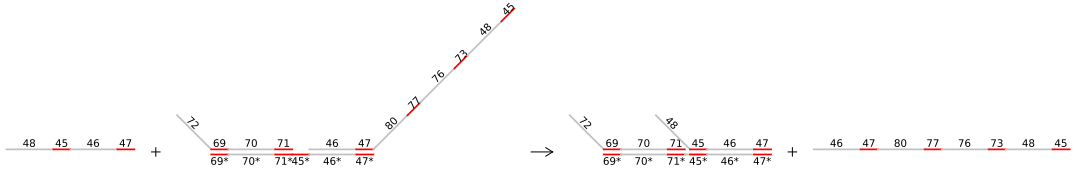
$$O_{R06} + T_{R06} \rightarrow WT2_{R06} + Y + X_2^L + W_2$$
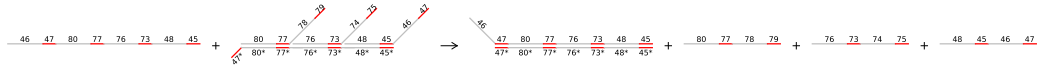
$$Y + L_{R07} \leftrightarrow H_{R07} + B_{R07}$$
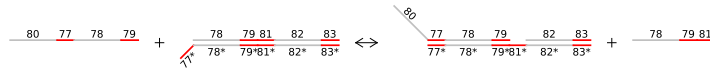
$$Y^{\ominus} + H_{R07} \rightarrow WT1_{R07} + O_{R07}$$
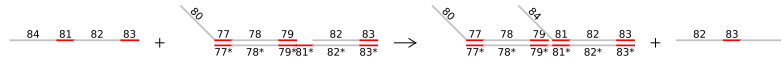
$$\hat{Y} + L_{R08} \leftrightarrow H_{R08} + B_{R08}$$
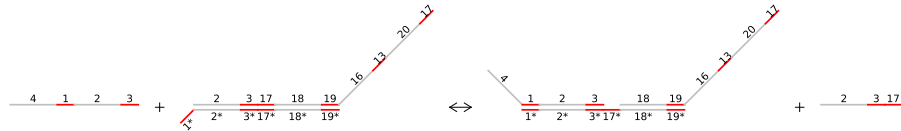
$$S_L + H_{R08} \rightarrow WT1_{R08} + O_{R08}$$

$$O_{R08} + T_{R08} \rightarrow WT2_{R08} + E^{\oplus} + S_L$$

$$Y + L_{R09} \leftrightarrow H_{R09} + B_{R09}$$

$$S_L + H_{R09} \rightarrow WT1_{R09} + O_{R09}$$

$$O_{R09} + T_{R09} \rightarrow WT2_{R09} + E^{\ominus} + S_L$$

$$E^{\oplus} + L_{R10} \leftrightarrow H_{R10} + B_{R10}$$

$$E_{decay} + H_{R10} \rightarrow WT1_{R10} + O_{R10}$$

$$O_{R10} + T_{R10} \rightarrow WT2_{R10} + E_{decay}$$

$$E^{\ominus} + L_{R11} \leftrightarrow H_{R11} + B_{R11}$$

$$E_{decay} + H_{R11} \rightarrow WT1_{R11} + O_{R11}$$

$$O_{R11} + T_{R11} \rightarrow WT2_{R11} + E_{decay}$$

$$E^{\oplus} + G_{R12} \rightarrow WT1_{R12} + O_{R12}$$

$$O_{R12} + T_{R12} \rightarrow WT2_{R12} + W^{\oplus} + E^{\oplus}$$

$$E^{\ominus} + G_{R13} \rightarrow WT1_{R13} + O_{R13}$$

$$O_{R13} + T_{R13} \rightarrow WT2_{R13} + W^{\ominus} + E^{\ominus}$$

$$W^{\oplus} + L_{R14} \leftrightarrow H_{R14} + B_{R14}$$

$$S^{L}_{in} + H_{R14} \rightarrow WT1_{R14} + O_{R14}$$

$$O_{R14} + T_{R14} \rightarrow WT2_{R14} + W_{0} + S^{L}_{in}$$

$$W^{\ominus} + L_{R15} \leftrightarrow H_{R15} + B_{R15}$$

$$S_{in}^L + H_{R15} \rightarrow WT1_{R15} + O_{R15}$$

$$O_{R15} + T_{R15} \rightarrow WT2_{R15} + W_0^\ominus + S_{in}^L$$
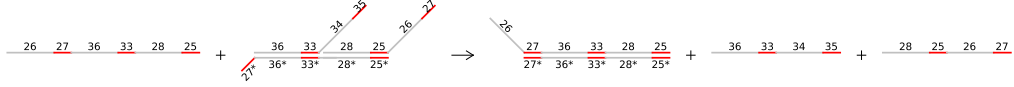
$$W_0 + L_{R16} \leftrightarrow H_{R16} + B_{R16}$$
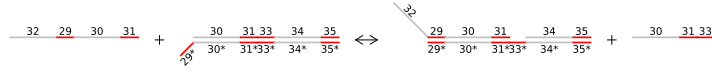
$$W_0^\ominus + H_{R16} \rightarrow WT1_{R16} + O_{R16}$$
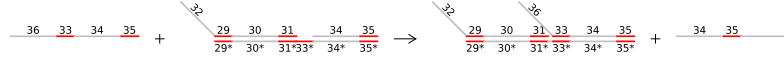
$$W^\oplus + L_{R17} \leftrightarrow H_{R17} + B_{R17}$$
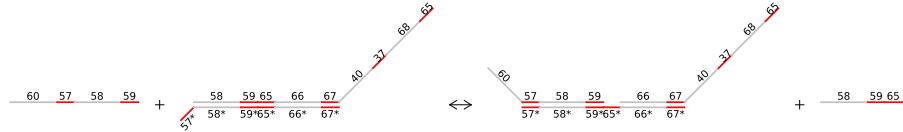
$$X_1^L + H_{R17} \rightarrow WT1_{R17} + O_{R17}$$
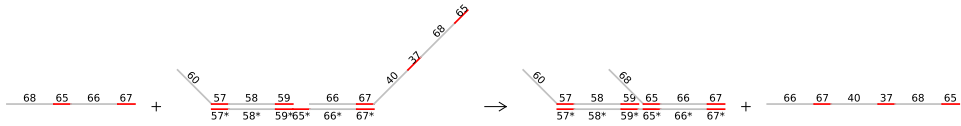
$$O_{R17} + T_{R17} \rightarrow WT2_{R17} + W_1 + X_1^L$$
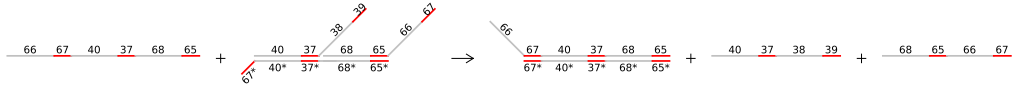
$$W^\ominus + L_{R18} \leftrightarrow H_{R18} + B_{R18}$$
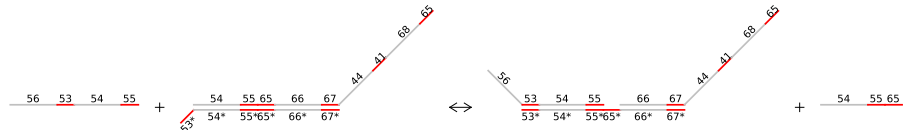
$$X_1^L + H_{R18} \rightarrow WT1_{R18} + O_{R18}$$
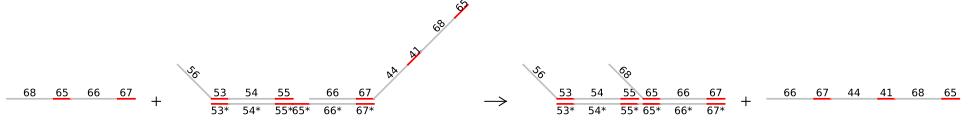
$$O_{R18} + T_{R18} \rightarrow WT2_{R18} + W_1^\ominus + X_1^L$$
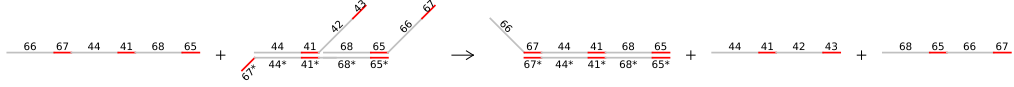
$$W_1 + L_{R19} \leftrightarrow H_{R19} + B_{R19}$$
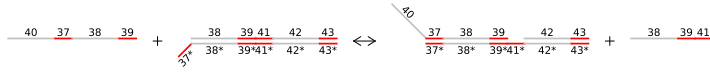
$$W_1^\ominus + H_{R19} \rightarrow WT1_{R19} + O_{R19}$$
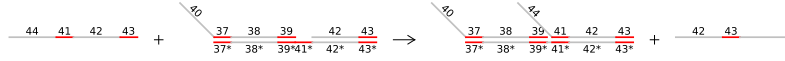
$$W^\oplus + L_{R20} \leftrightarrow H_{R20} + B_{R20}$$
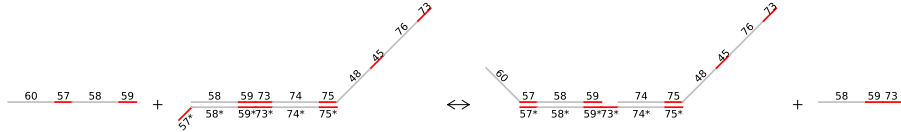
$$X_2^L + H_{R20} \rightarrow WT1_{R20} + O_{R20}$$
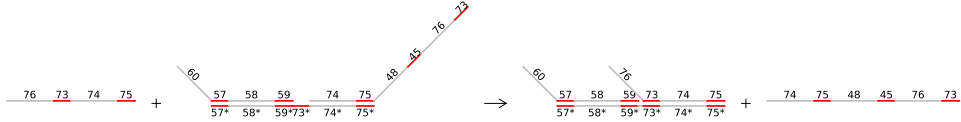
$$O_{R20} + T_{R20} \rightarrow WT2_{R20} + W_2 + X_2^L$$
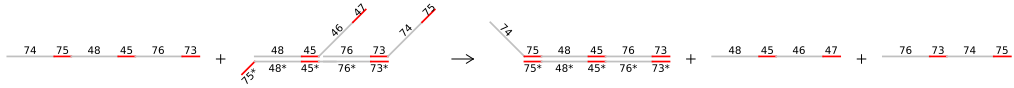
$$W^\ominus + L_{R21} \leftrightarrow H_{R21} + B_{R21}$$
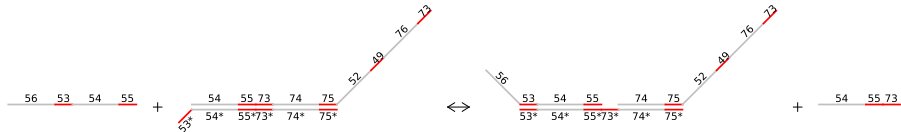
$$X_2^L + H_{R21} \rightarrow WT1_{R21} + O_{R21}$$

$$O_{R21} + T_{R21} \rightarrow WT2_{R21} + W_2^{\ominus} + X_2^L$$

$$W_2 + L_{R22} \leftrightarrow H_{R22} + B_{R22}$$

$$W_2^{\ominus} + H_{R22} \rightarrow WT1_{R22} + O_{R22}$$

$$S_{in} + LS_{S_{in}} \leftrightarrow HS_{S_{in}} + BS_{S_{in}}$$

$$X_1 + LS_{X_1} \leftrightarrow HS_{X_1} + BS_{X_1}$$

$$X_2 + LS_{X_2} \leftrightarrow HS_{X_2} + BS_{X_2}$$

$$W_0 + LS_{W_0} \leftrightarrow HS_{W_0} + BS_{W_0}$$

$$W_1 + LS_{W_1} \leftrightarrow HS_{W_1} + BS_{W_1}$$

$$W_2 + LS_{W_2} \leftrightarrow HS_{W_2} + BS_{W_2}$$

$$S_{in}^L + LS_{S_{in}^L} \leftrightarrow HS_{S_{in}^L} + BS_{S_{in}^L}$$

$$X_1^L + LS_{X_1^L} \leftrightarrow HS_{X_1^L} + BS_{X_1^L}$$

$$X_2^L + LS_{X_2^L} \leftrightarrow HS_{X_2^L} + BS_{X_2^L}$$

76 73 74 75 + 74 75101 / 73* 74* 75101* ↔ 76 / 73 74 75 / 73* 74* 75101* + 74 75101

$$E^\ominus + LS_{E^\ominus} \leftrightarrow HS_{E^\ominus} + BS_{E^\ominus}$$

12 9 10 11 + 10 11104 / 9* 10* 11104* ↔ 12 / 9 10 11 / 9* 10* 11104* + 10 11104

$$E^\oplus + LS_{E^\oplus} \leftrightarrow HS_{E^\oplus} + BS_{E^\oplus}$$

16 13 14 15 + 14 15102 / 13* 14* 15102* ↔ 16 / 13 14 15 / 13* 14* 15102* + 14 15102

$$W^\ominus + LS_{W^\ominus} \leftrightarrow HS_{W^\ominus} + BS_{W^\ominus}$$

56 53 54 55 + 54 55 99 / 53* 54* 55*99* ↔ 56 / 53 54 55 / 53* 54* 55*99* + 54 55 99

$$W^\oplus + LS_{W^\oplus} \leftrightarrow HS_{W^\oplus} + BS_{W^\oplus}$$

60 57 58 59 + 58 59 97 / 57* 58* 59*97* ↔ 60 / 57 58 59 / 57* 58* 59*97* + 58 59 97

$$W_0^\ominus + LS_{W0^\ominus} \leftrightarrow HS_{W0^\ominus} + BS_{W_0^\ominus}$$

36 33 34 35 + 34 35 92 / 33* 34* 35*92* ↔ 36 / 33 34 35 / 33* 34* 35*92* + 34 35 92

$$W_1^\ominus + LS_{W_1^\ominus} \leftrightarrow HS_{W_1^\ominus} + BS_{W_1^\ominus}$$

44 41 42 43 + 42 43 93 / 41* 42* 43*93* ↔ 44 / 41 42 43 / 41* 42* 43*93* + 42 43 93

$$W_2^\ominus + LS_{W_2^\ominus} \leftrightarrow HS_{W_2^\ominus} + BS_{W_2^\ominus}$$

52 49 50 51 + 50 51 94 / 49* 50* 51*94* ↔ 52 / 49 50 51 / 49* 50* 51*94* + 50 51 94

$$S_L + LS_{S_L} \leftrightarrow HS_{S_L} + BS_{S_L}$$

20 17 18 19 + 18 19 96 / 17* 18* 19*96* ↔ 20 / 17 18 19 / 17* 18* 19*96* + 18 19 96

$$Y^\ominus + LS_{Y^\ominus} \leftrightarrow HS_{Y^\ominus} + BS_{Y^\ominus}$$

84 81 82 83 + 82 83 91 / 81* 82* 83*91* ↔ 84 / 81 82 83 / 81* 82* 83*91* + 82 83 91

$$\hat{Y} + LS_{\hat{Y}} \leftrightarrow HS_{\hat{Y}} + BS_{\hat{Y}}$$

4 1 2 3 + 2 3 95 / 1* 2* 3*95* ↔ 4 / 1 2 3 / 1* 2* 3*95* + 2 3 95

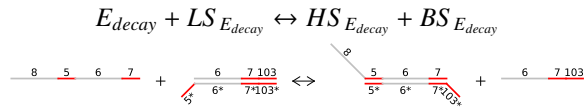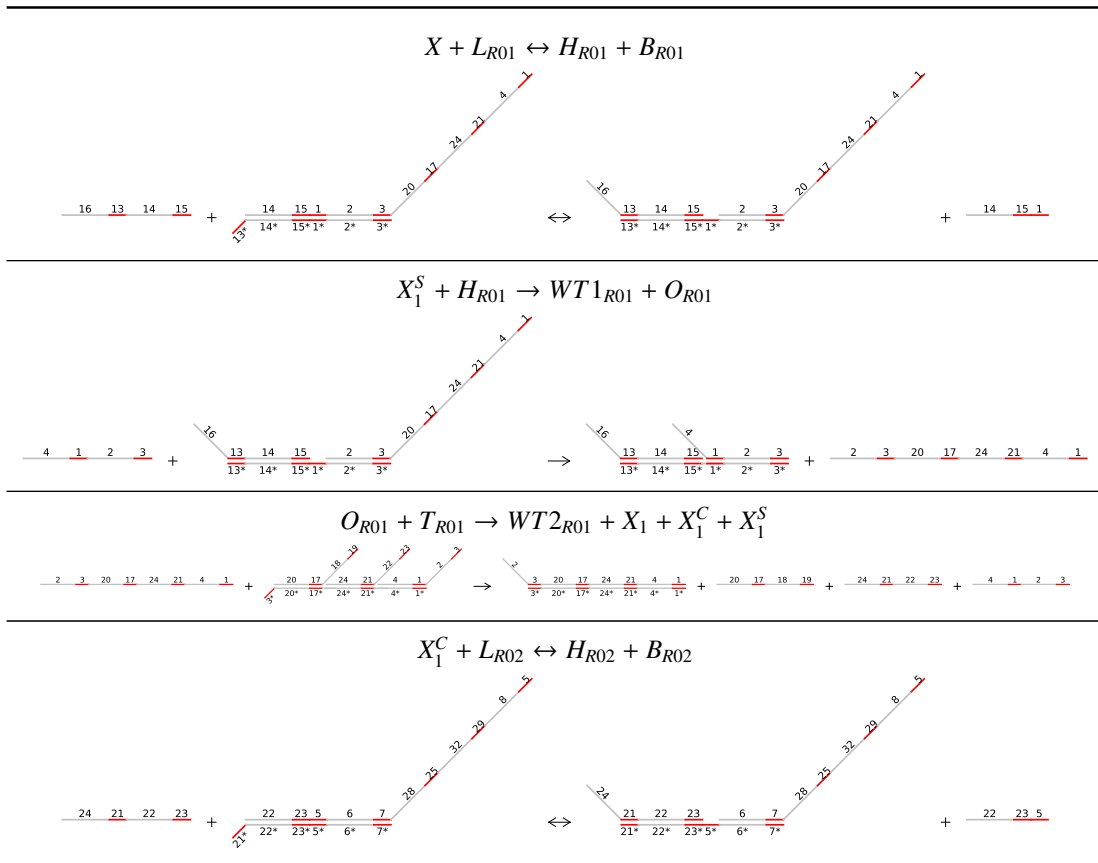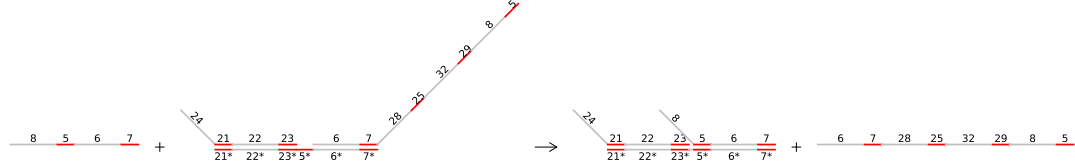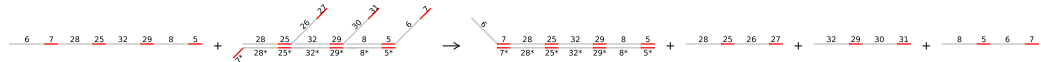$$E_{decay} + LS_{E_{decay}} \leftrightarrow HS_{E_{decay}} + BS_{E_{decay}}$$

Figure C.1: Full list of domain-specified DNA strand displacement reactions implementing the linear chemical perceptron.

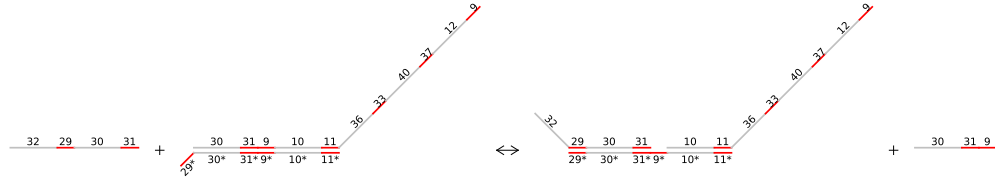$$X + L_{R01} \leftrightarrow H_{R01} + B_{R01}$$

$$X_1^S + H_{R01} \rightarrow WT1_{R01} + O_{R01}$$

$$O_{R01} + T_{R01} \rightarrow WT2_{R01} + X_1 + X_1^C + X_1^S$$

$$X_1^C + L_{R02} \leftrightarrow H_{R02} + B_{R02}$$

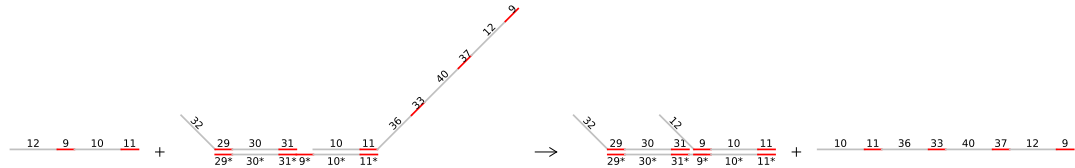$$X_2^S + H_{R02} \rightarrow WT1_{R02} + O_{R02}$$

$$O_{R02} + T_{R02} \rightarrow WT2_{R02} + X_2 + X_2^C + X_2^S$$
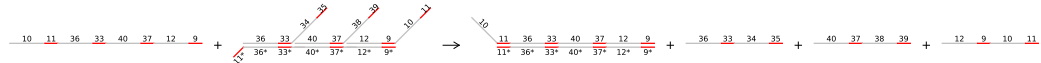

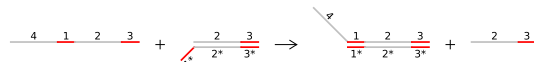$$X_2^C + L_{R03} \leftrightarrow H_{R03} + B_{R03}$$

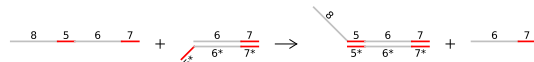$$X_3^S + H_{R03} \rightarrow WT1_{R03} + O_{R03}$$

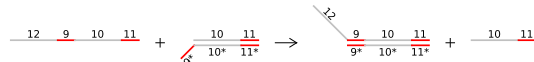$$O_{R03} + T_{R03} \rightarrow WT2_{R03} + X_3 + X_3^C + X_3^S$$

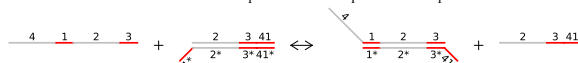$$X_1^S + G_{R04} \rightarrow WT1_{R04} + O_{R04}$$

$$X_2^S + G_{R05} \rightarrow WT1_{R05} + O_{R05}$$

$$X_3^S + G_{R06} \rightarrow WT1_{R06} + O_{R06}$$

$$X_1^S + LS_{X_1^S} \leftrightarrow HS_{X_1^S} + BS_{X_1^S}$$

$$X_2^S + LS_{X_2^S} \leftrightarrow HS_{X_2^S} + BS_{X_2^S}$$

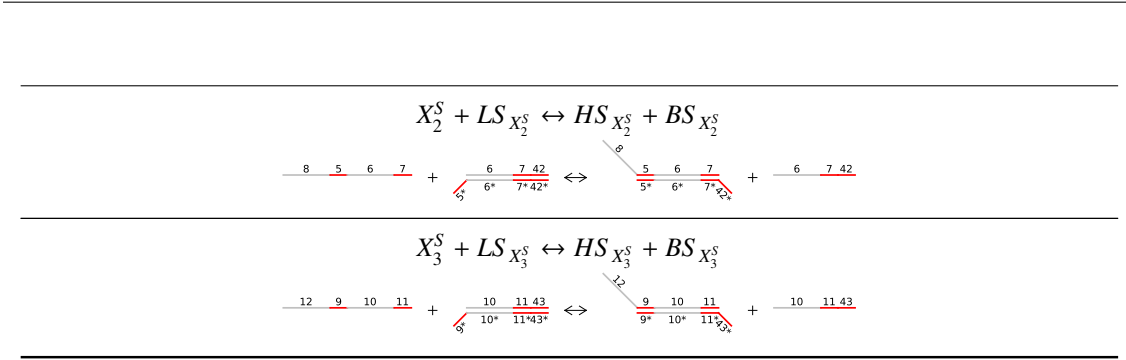$$X_3^S + LS_{X_3^S} \leftrightarrow HS_{X_3^S} + BS_{X_3^S}$$

Figure C.2: Full list of domain-specified DNA strand displacement reactions implementing the manual signalling delay line of size three.

Appendix D

VARIOUS DATA

Table D.1: The bounds that restrict the value range of rate constants during mutation, and the generation of initial population in GA. They are specified for two reaction types: mass-action and catalysis.

Type	Reaction	Rate	Rate Constant Bounds
Mass-action	$S \rightarrow P$	$k[S]$	$k \in [0, 0.6]$
Catalysis	$E + S \rightleftharpoons ES \rightarrow E + P$	$\frac{k_{cat}[E][S]}{K_m + [S]}$	$k_{cat} \in [0, 0.6], K_m \in [0, 5]$

Table D.2: The GA setting and parameter values.

Attribute	Value
Selection type	one-point
Population size	100
Elite size	20
Generation limit	50
Cross-over probability	0.9
Per-element mutation probability	0.3
Uniform mutation strength	0.3

Name	f(1,1)	f(1,0)	f(0,1)	f(0,0)
FALSE	0	0	0	0
NOR	0	0	0	1
NCIMPL	0	0	1	0
NOT X1	0	0	1	1
NIMPL	0	1	0	0
NOT X2	0	1	0	1
XOR	0	1	1	0
NAND	0	1	1	1
AND	1	0	0	0
XNOR	1	0	0	1
PROJ X2	1	0	1	0
IMPL	1	0	1	1
PROJ X1	1	1	0	0
CIMPL	1	1	0	1
OR	1	1	1	0
TRUE	1	1	1	1

Figure D.1: All 2-input logic functions with associated truth (output) tables.