



HAL
open science

Biomolecular System Design: Architecture, Synthesis, and Simulation

Katherine Chiang

► **To cite this version:**

Katherine Chiang. Biomolecular System Design: Architecture, Synthesis, and Simulation. Programming Languages [cs.PL]. National Taiwan University, 2015. English. NNT: . tel-01237638

HAL Id: tel-01237638

<https://inria.hal.science/tel-01237638>

Submitted on 3 Dec 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

國立臺灣大學電機資訊學院電子工程學研究所
博士論文

Department or Graduate Institute of Electronics Engineering
College of Electrical Engineering and Computer Science
National Taiwan University
Doctoral Dissertation

生物分子計算系統設計：架構、合成、與模擬

Biomolecular System Design:
Architecture, Synthesis, and Simulation

姜慧如

Katherine H. Chiang

指導教授 (Advisors):

江介宏 博士 (Jie-Hong R. Jiang, Ph.D.)

François Fages, Ph.D.

中華民國 104 年 6 月

June 2015

Biomolecular System Design: Architecture, Synthesis, and Simulation

By

Katherine H. Chiang

Advisors: Dr. Jie-Hong R. Jiang and Dr. François Fages

Graduate Institute of Electronics Engineering
National Taiwan University

Abstract

The advancements in systems and synthetic biology have been broadening the range of realizable systems with increasing complexity both *in vitro* and *in vivo*. Systems for digital logic operations, signal processing, analog computation, program flow control, as well as those composed of different functions – for example an on-site diagnostic system based on multiple biomarker measurements and signal processing – have been realized successfully. However, the efforts to date tend to tackle each design problem separately, relying on *ad hoc* strategies rather than providing more general solutions based on a *unified* and *extensible* architecture, resulting in long development cycle and rigid systems that require redesign even for small specification changes.

Inspired by well-tested techniques adopted in electronics design automation (EDA), this work aims to remedy current design methodology by establishing a standardized, complete flow for realizing biomolecular systems. Given a behavior specification, the flow streamlines all the steps from modeling, synthesis, simulation, to final technology mapping onto implementing chassis. The resulted biomolecular systems of our design flow are all built on top of an FPGA-like *reconfigurable* architecture with recurring *modules*. Each module is designed the function of each

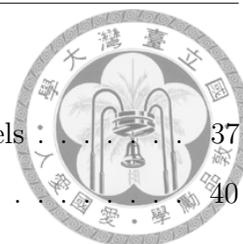
module depends on the concentrations of assigned auxiliary species acting as the “tuning knobs.” Reconfigurability not only simplifies redesign for altered specification or post-simulation correction, but also makes post-manufacture fine-tuning – even after system deployment – possible. This flexibility is especially important in synthetic biology due to the unavoidable variations in both the deployed biological environment and the biomolecular reactions forming the designed system.

In fact, by combining the system’s reconfigurability and neural network’s self-adaptiveness through learning, we further demonstrate the high compatibility of *neuromorphic computation* to our proposed architecture. Simulation results verified that with each module implementing a neuron of selected model (ex. spike-based, threshold-gate-like, etc.), accompanied by an appropriate choice of reconfigurable properties (ex. threshold value, synaptic weight, etc.), the system built from our proposed flow can indeed perform desired neuromorphic functions.



Contents

1	Introduction	1
1.1	Motivation	1
1.2	Our contributions	2
1.3	Structure of the thesis	4
2	Hybrid simulation of heterogeneous reaction models	7
2.1	SBML reaction rules and events	12
2.1.1	Reaction rules and kinetics	12
2.1.2	Events	14
2.2	Hybrid continuous-stochastic models	17
2.2.1	Gillespie's stochastic simulation algorithm (SSA)	18
2.2.2	Event-based implementation of SSA	19
2.2.3	Preprocessor for composing continuous and stochastic models	21
2.3	Dynamic strategies for hybrid continuous-stochastic simulations . . .	24
2.3.1	Dynamic partitioning criteria	25
2.3.2	Implementation	26
2.3.3	Simple example	28
2.3.4	Performance evaluation	30
2.4	Hybrid Boolean models	35
2.4.1	Preprocessor for composing continuous and Boolean models .	35



2.4.2	Hybrid composition of continuous-Boolean models	37
2.4.3	Stochastic-continuous-Boolean model simulation	40
3	Reconfigurable biochemical reaction systems synthesis	41
3.1	Modeling language	41
3.2	Reconfigurable logic circuit synthesis	42
3.2.1	Configurable logic units	44
3.2.2	Configurable interconnects	48
3.2.3	Retroactivity resistance	49
3.2.4	Multiplexer-based structure	51
3.2.5	Case study	54
3.3	Level-based neuromorphic computation	58
3.3.1	Artificial neural network and adopted neuron model	61
3.3.2	General architecture of neuromorphic FPGA	63
3.3.3	Neuron module	66
3.3.4	Programmable interconnect	70
3.3.5	Resource requirement	71
3.3.6	Case study: classifier synthesis with known criteria	71
3.3.7	Case study: classifier synthesis with learning ability	75
3.4	Spike-based neuromorphic computation	77
3.4.1	Adopted neuron model and key properties	79
3.4.2	Directional signal transmission with waveform preserved	81
3.4.3	Neuron module	82
4	Technology mapping and implementation	85
4.1	Enzyme	86
4.1.1	Enzyme kinetics	91



4.1.2	Reaction motifs	94
4.1.3	Mapping example: configurable Boolean logic gate	95
4.2	DNA strand displacement	100
4.2.1	DSD reaction modules as mapping target	103
4.2.2	Reaction rate	103
4.2.3	Mapping example: circuit	103
4.2.4	Mapping example: classifier	104
5	Conclusions and future work	112
A	DSD simulation code	113
A.1	Code for mapping example: circuit	113
A.2	Code for mapping example: classifier	117
B	Hudgkin-Huxley model simulation	122
B.1	Code for mapping example: circuit	122
	List of Figures	129
	List of Tables	131
	Bibliography	142



Chapter 1

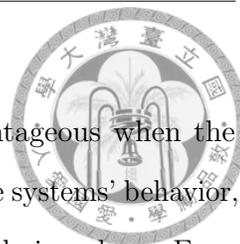
Introduction

1.1 Motivation

The advancements of synthetic biology have been broadening the range of realizable systems of increasing complexity both *in vivo* and *in vitro*. Building systems within a biochemical world is not far from reach and has been intensively studied, e.g., in terms of digital logic operations [37, 47, 63], analog computation [20], linear control [17, 60], signal processing [48], program flow control [44], etc. The biocompatibility of such systems is unique in that they can not only allow effective *interfacing* between physiological processes and nano-structured materials as well as electronic systems, but can further embed computation tasks that integrate sensing, information processing, and actuation, inside living cells without physical intrusion.

However, most, if not all, of existing engineered biochemical systems perform specific functions with fixed parametric values, which once designed, cannot be changed. This static approach toward functions and parameters is preferred for

1.2. Our contributions



design analysis and verification, but can be seriously disadvantageous when the uncertainty involved in environmental evolution is influential to the systems' behavior, or when the systems' behavior *cannot* be fully determined in the design phase. Even for electronic system design—which is relatively very predictable as hinted by the existence of clearly defined datasheets—it is still not uncommon that a design has to be rectified *after* it is manufactured. Likewise in biochemical system design, *reconfigurability* is beneficial and usually *crucial* because of the intrinsically stochastic biochemical environments. While the reconfigurability of integrated circuits (ICs) can be achieved through embedding firmware or programmable gate arrays into the design, it remains unclear how a similar mechanism can be economically embedded into a biochemical design.

Inspired by the success achieved by *electronics design automation (EDA)* methodology in efficiently realizing systems of fast-growing complexity, flexibility, and robustness, we tried to address this deficiency by proposing a new synthetic biology design framework with lessons learned from EDA to deal with shared concerns, while also modified to take into account the fundamental differences of the two engineering paradigms.

1.2 Our contributions

The proposed framework is based on a flexible, modular architecture that is similar to a field programmable gate array (FPGA). Specifically, emphasis is placed on the resulted systems' *reconfigurability*, which is of crucial importance for system reliability in the biochemical world, where variations of common degree can cause serious functional deviations in the deployed systems. In this work, the choice of

1.2. Our contributions



adopting *chemical reaction network (CRN)* as the middle *description* language for bridging the user specifications to kinetics desired of molecular interactions is not only based on the fact proved in [72] that CRNs form a Turing-universal computational model capable of encoding any kind of computational behavior, but also on its being standardized and well-accepted across various disciplines.

In the proposed design framework, a flow for realizing biomolecular systems as well as its accompanying *reservoirs* of standardized modules (motifs) are constructed. Given a behavior specification, the proposed framework synthesizes the specification to a CRN that is amenable to final implementations using enzyme reactions and/or DNA strand displacement. The flow streamlines all steps from processing system behavior specification until right before where the wetlab experiments would join in the future— from modeling, synthesis, simulation, to final technology mapping onto implementing chassis of choice. The accompanying reservoirs of reconfigurable modules are designed with loading effects (retroactivities) considered. Different reservoirs of modules are prepared not only for different classes of specialized functionalities (ex. Boolean logic operations or neuromorphic computations), but also for different technology mapping targets—we try to have the structure of chemical reaction network (CRN) in the modules more similar to that of the existing ones presented in targeted chassis—in order to increase the probability that the modules can be more directly mapped to existing reactions.

Apart from taking care of the above considerations, two other novel extensions are made to the framework:

First, multiple kinds of system specification are allowed to accommodate the richness and uncertainty of biochemical applications. Current available options include:

- Boolean logic formula as in combinational circuit design;

1.3. Structure of the thesis



- neuromorphic algorithm with interconnects and update functions specified;
- training datasets based on which the system would learn the unknown “correct” function, and would automatically evolve to realize the function learned.

Second, this work takes an important step toward embedding the power of neural networks into a broader range of biological systems. A correspondence between the information transmission mechanisms of ubiquitous cell signaling pathways and the action potential propagation in neurons of Hodgkin-Huxley model is established. Based on the correspondence, we propose an analog approach to realize reconfigurable neuromorphic computation using existing biochemical reactions of signaling pathways for better bio-compatibility.

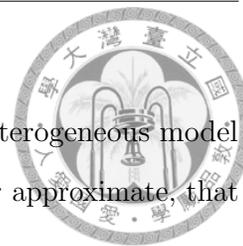
1.3 Structure of the thesis

This thesis is organized as follows:

- **Chapter 2** shows a new, nonstandard way to define stochastic and Boolean simulators by properly combining reaction rules and events, justifying the hybrid composition and simulation of heterogeneous biochemical reaction models. Specifically, a high-level interface is implemented to show how two SBML models of different interpretations can be effectively composed into one. Furthermore, dynamic partitioning strategies for automatically partitioning reactions into stochastic or continuous interpretations based on adaptive criteria are presented, with gain in both accuracy and simulation time compared to static partitioning.

Section 2.1 first gives a review on the SBML definition of reaction *rules* and

1.3. Structure of the thesis



events, which will serve as the two components for our heterogeneous model construction, and whose kinetics will be used to capture, or approximate, that of the modeled system for hybrid simulation.

Section 2.2 presents the non-standard way of using *event* to embed stochastic reaction occurrences into continuous system evolution. A proof-of-concept implementation of a preprocessor that can automatically generate reaction-event-based hybrid models is presented in the end.

In **Section 2.3**, considering the fact that all interpretations have their respective applicable conditions, strategies for dynamically adjusting the interpretations of the reactions during system evolution are presented and implemented to more accurately capture the behavior under live condition.

Section 2.4 focuses on hybrid model construction with Boolean semantics involved. Boolean semantics provides an effective way to capture switch-like behavior as that demonstrated in genetic networks, and a natural interface with finite state machine.

- **Chapter 3** is devoted to the biochemical reaction synthesis of our proposed reconfigurable architectures. Different specification formats are provided so that system requirements can be specified in formats that can accurately portrait the desired behaviors in a convenient way. Firstly in **Section 3.1**, the motivation of our choice of using chemical reaction network (CRN) as the description language is given. CRN forms the bridge between behavior specification to required kinetics in the sense that once fulfilled, the desired behavior can be realized.

The synthesis of reconfigurable logic specification is presented in **Section 3.2**. The system is based on reconfigurable logic gates whose functions can be switched between a set of *functional complete* logic operations simply by

1.3. Structure of the thesis



concentration controls. The architecture is very similar to the FPGA, except that the species used for different gates should not introduce “undesigned” interactions.

In **Section 3.3** and **Section 3.4**, the synthesis of neuromorphic computation based on threshold gate-like binary neuron model and spike-timing based neuron model are presented respectively.

- The networks of reactions synthesized as described in Chapter 3 all have CRN-modeled behavior that, *theoretically*, satisfy the specified requirements. **Chapter 4** shows the preliminary attempt to map the synthesized designs to realistic biochemical reactions based on real-world motifs extracted from existing reactions of targeted chassis.

The mapping to two kinds of targeted chassis are discussed. In **Section 4.1**, the enzyme realization of kinetics described by CRNs is presented along with the kinetics motifs based on six standard types of enzyme-catalyzed reactions. In **Section 4.2**, we go through the well-established mapping relations between CRN and DNA strand displacement kinetics, and used Visual DSD simulation of the accordingly mapped strands as the verification of our proposed synthesis method.

- Finally, **Chapter 5** concludes this work and outlines possible directions for future work.



Chapter 2

Hybrid simulation of heterogeneous reaction models

Systems biology aims to elucidate the high-level functions of the cell from their biochemical basis at the molecular level [46]. A lot of work has been done for collecting genomic and post-genomic data, making them available in databases [6, 49], and organizing the knowledge on pathways and interaction networks into models of cell metabolism, signaling, cell cycle, apoptosis, etc., many of which are published in model repositories such as <http://biomodels.net/>. Aiding the efforts, the Systems Biology Markup Language (SBML) [45] provides a common exchange format for biochemical *reaction systems* and is nowadays supported by a majority of modeling tools.

According to the knowledge available on the system and to the nature of queries expected to be answered by the model, e.g. qualitative or quantitative predictions, these rule-based reaction systems can be interpreted (and simulated) under different semantics as either:



- ordinary differential equations (continuous semantics),
- continuous-time Markov chains (stochastic semantics),
- Petri nets (discrete semantics),
- Boolean transition systems (Boolean semantics), and many variants.

These different interpretations can be related by either approximation [30–32] or abstraction [23] relationships. Many modeling tools support several of them but provide no support for the *combination* of heterogeneous models. However, in the perspective of applying engineering methods to the analysis and control of biological systems, the issue of building complex models by composition of elementary models is a central one. While reaction systems can be formally composed by the multiset union of reaction rules, and interpreted by one common semantics, there is also a need to compose models with different semantics preserved, as will be clearly shown in this section by some examples from the literature. What we call a *hybrid model* is a model obtained by composing models of heterogeneous semantics (continuous, stochastic, Boolean, etc.), and *hybrid simulation* is the topic of simulating such hybrid models.

In [61], the author observes that “A very promising direction is the development of hybrid methods because they directly deal with the important problem of stiffness, which is often present in biochemical models. [...] There exist already a few software tools, which allow for hybrid simulation, [...] and this number is expected to grow in the future.” In this chapter, we propose a general approach to progress in that direction by showing that the combination of reaction rules and events, as already present in SBML, can be used in a non-standard way to give meaning to the hybrid composition and simulation of heterogeneous reaction models. In particular, we show



how hybrid continuous-Boolean models and hybrid continuous-stochastic models can be assembled and simulated, through the specification of a *high-level interface for composing heterogeneous models*, and producing as output a hybrid model in standard SBML format, which can thus be executed with any SBML-compatible simulator.

Our high-level interface, prototyped in the modeling environment BIOCHAM [13, 24], takes two models with synchronization information as inputs, and produces one SBML model with reactions and events as output. For hybrid continuous-Boolean composition, it transforms a Boolean state transition model to events with extra triggers which express the links with the continuous variables and the parameters of the continuous reaction model. For hybrid continuous-stochastic composition, the interface described in this chapter transforms stochastic reactions to a set of events, which implements Gillespie's direct method for stochastic simulation, and can be freely combined with the simulation of continuous reactions. Furthermore, our framework supports the specification of *dynamic strategies* which automatically choose between the stochastic and continuous interpretations of the reactions according to particle counts, reaction propensities, or more specific model-dependent criteria. We show that without the need to conduct time-consuming fully stochastic simulations beforehand to obtain the scale information of particle count and propensity for all reactions, dynamic partitioning results in higher accuracy and shorter simulation time than static partitioning – as static partitioning cannot adapt to the possibly substantial scale variations over time, which can render the initial partition inadequate.

This approach is illustrated and evaluated with several examples including the reconstructions of the hybrid model of the mammalian cell cycle regulation of Singhania et al. [71] as the composition of a Boolean model of cell cycle phase



transitions and a continuous model of cyclin activation, plus a hybrid Boolean-continuous-stochastic version of this model with dynamic partitioning strategy, and of the hybrid stochastic-continuous model of bacteriophage T7 infection of Alfonsi et al. [4], and of bacteriophage λ of Goutsias [40], showing the gain in both accuracy and simulation time of the dynamic strategy.

Since XML, and hence SBML, are not easy to read by humans, in this thesis, we use mathematical notation and BIOCHAM code for clarity purpose. The BIOCHAM and SBML files of the examples of this chapter are available at: http://lifeware.inria.fr/supplementary_material/TOMACS/. The BIOCHAM files can be executed via the BIOCHAM web application <http://lifeware.inria.fr/biocham/online> without any installation.

Related work

Hybrid simulation is a classical topic in physics, e.g. for numerically solving equations describing stochastic systems using ordinary differential equations whenever possible in place of stochastic equations, in order to speed-up simulations [4, 67]. It is also ubiquitous in computer science for programming and verifying hybrid systems which have both discrete and continuous dynamics [5, 39]. Hybrid modeling is also used in Systems Biology for reducing the complexity of many modeling task, e.g. [1, 5, 9, 11, 27, 51, 58, 71], for speeding up stochastic simulations [33, 36, 40, 68], and achieving whole cell simulation [50]. A review of the different approximate stochastic and hybrid methods used in Systems Biology can be found in [61].

Due to the structure of SBML, which mostly relies on explicit and global reactions and events, the composable modelling at the core of hybrid process algebra, e.g. [3, 26]



is out of reach of the presented work. On the other hand, we show that SBML can express various form of hybrid systems. Indeed, a set of SBML events and continuous reactions can also be visualized as a hybrid automaton [38] in which there is a state with a particular ODE for each combination of the trigger values, and there is a transition from one state to another state when at least one trigger changes value from false to true in the source state. Stochastic hybrid automata [34] can be similarly simulated in SBML with a random number generator coded by events. Since our focus in this chapter is on SBML, we are mainly focused on simulations and on the reproduction of simulation results, as exemplified for instance by the notion of “curated” model in the BIOMODELS project at <http://biomodels.net/>. The use of existing verification tools for hybrid systems is thus beyond the scope of this thesis.

Another line of work also exists on the extension of Boolean models with continuous time delays. René Thomas’s discrete modeling of gene regulatory networks (GRN) [73] is a well known approach to study the logical dynamics of a set of interacting genes. It deals with a graph of positive and negative influences between genes and logical functions that determine the possible trajectories in the state space. Those parameters are a priori unknown, but they may generally be deduced from a large set of biologically observed behaviors in various conditions. Besides, it neglects the time delays for a gene to pass from one level of expression to another one. In [1], it is shown that one can account for time delays depending on the expression levels of genes in a GRN, while preserving powerful enough computer-aided reasoning capabilities. The characteristic of this approach is that, among possible execution trajectories in the model, one can automatically find out both viability cycles and absorption in capture basins. Model-checking techniques developed for hybrid systems are used for this purpose [2]. The authors describe a Hybrid model for the mucus

2.1. SBML reaction rules and events



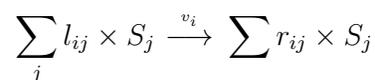
production in the bacterium *Pseudomonas aeruginosa* and show that they are able to discriminate between various possible dynamical behavior [1,2]. Such a model can be presented and compiled in a set of reaction rules with events as described in this thesis.

Time constraints provide another means to refine Boolean or discrete models which are often too coarse to be useful. In [56], the authors present a new technique for over-approximating (in the sense of timed trace inclusion) continuous dynamical systems by timed automata for the purpose of efficiently checking timed (as well as untimed) properties. The essence of this technique is the partition of the state space into cubes and the allocation of a clock for each dimension. This is in contrast with other approaches which use only one clock. This idea is a specific case of rectangular hybrid automata. This makes it possible to get better approximations of the behavior. The timed automata produced by these techniques can be similarly composed in our tool for simulation.

2.1 SBML reaction rules and events

2.1.1 Reaction rules and kinetics

In SBML [45], a reaction rule is composed of a reaction rate, a left- and a right-hand side of molecular species, with corresponding stoichiometric coefficients. In this thesis, an SBML reaction i written in mathematical notation:



2.1. SBML reaction rules and events



corresponds to the following code in BIOCHAM syntax:

$$v_i \text{ for } \sum_j l_{ij} * S_j \Rightarrow \sum_j r_{ij} * S_j$$

where v_i is any mathematical function (given in a subset of MathML notation in SBML, and in BIOCHAM with the abbreviation MA for mass action law kinetics) of the species concentrations and parameters of the system, which defines the rate of reaction i . A *reaction model* is a finite set of reactions.

Depending on the data available of the system and the nature of queries that the model aims to address to, e.g. qualitative or quantitative predictions, a reaction model can be interpreted under different semantics: continuous, stochastic, discrete or boolean.

The *stochastic* semantics, which would be detailed in Section 2.2.1, associates to a reaction model a *Continuous-Time Markov Chain (CTMC)*, with states defined by molecular counts, transitions by reactions, and reaction rates equal propensities representing transition probabilities after normalization. The associated CTMC realizes the solution to the Chemical Master Equation [30] of the reaction model.

The *continuous* semantics associates to a reaction model an Ordinary Differential Equation (ODE) system of the following form:

$$\frac{d[S_j]}{dt} = \sum_i (r_{ij} - l_{ij}) \times v_i$$

The ODE system describes the evolution of molecular species concentrations with time according to the reaction rates. The continuous semantics approximates the mean behavior of the CTMC for large numbers of molecules. The continuous semantics usually leads to numerical integration, whereas the stochastic semantics is either

2.1. SBML reaction rules and events



used for exact or approximate simulation, or for stochastic model checking (see for instance [52]).

The *discrete* semantics of a reaction model can be formalized using a Petri net [28], which keeps the stoichiometric information but leaves out the reaction rates v_i . The Petri net semantics can be seen as an abstraction of the stochastic semantics.

The *Boolean* semantics omits precise stoichiometry and keeps only information about whether or not a species is active. It can be defined as an abstraction of the previous discrete semantics, provided that the combinatorics of all possible consumptions is maintained [23]. The Boolean semantics of large networks is especially useful for efficiently proving reachability properties by symbolic model checkers [14] instead of directly performing simulations.

2.1.2 Events

SBML models also allow behavior description using *events*. An SBML event is basically composed of two parts: a *trigger* that specifies the condition for it to fire, and an *action* that describes its influence on current state (defined by concentrations and parameter values) in the form of a list of assignments. In this thesis, we will write an event in BIOCHAM syntax as follows:

$$event(trigger, [s_1, \dots, s_n], [f_1, \dots, f_n])$$

where s_i 's indicate the variables that are modified by the event, and f_i 's the mathematical functions of the state variables that give the new value to s_i 's.

There exists many possible semantics for interpreting events, but the central

2.1. SBML reaction rules and events



concept is the same: that an event fires when its triggering condition changes from *false* to *true*. This induces however several issues:

- what happens at the start of the simulation?
- how to determine the precise timing when a trigger condition turns to true?
- what happens if more than one events are enabled simultaneously?

The first point is easy to settle. Whether events that are true at time 0 should fire or not is an arbitrary choice with no impacts on the expressive power of the formalism, since the influence of both choices can be absorbed into the setting of initial state. In the BIOCHAM simulator used in this thesis, the choice is to avoid the firing of events at the initial point of the simulation; an event only gets triggered when its condition turns from false to true, i.e., it is the transition that matters. If an event is really meant to be triggered at the very beginning, the initial state is modified accordingly to reflect the changes in state variables caused by one firing of the event.

The second point, in fact, has been solved in practical tools for a long time: since numerical integration of ODEs goes by steps, one detects changes in triggers only in the interval of a simulation step. If some triggers become true, one can thus go back in time until one finds—with a given precision—the first time point where the first trigger becomes true. Note however that if arbitrarily complex conditions appear in the events, a numerical integrator unaware of the events can hide inside a single step that a trigger went from false to true and back to false again. Therefore, a cautious implementation is necessary, and fixed step size integration methods may be recommended to use in presence of events, instead of more efficient adaptive step size methods.

The third point is again a question with multiple possible answers. Generally, the

2.1. SBML reaction rules and events



set of events that are enabled simultaneously at a given time will all be fired, whatever the actions of the events are, but what if several events modify the same variable? It is possible to assume a *synchronous* semantics, where the simultaneous events execute their actions in parallel, but then one must forbid events with conflicting actions, i.e., events that would modify in different ways the same variable at the same time point. A more common choice is an *asynchronous* semantics, that will fire all the events enabled at a given time one after the other. Conflicts in actions are then solved by the ordering of events, which can be either random, i.e. non-deterministic, or given by the user, e.g. by the order of writing (BIOCHAM choice) or by priorities (SBML choice). However, if some actions invalidate the triggering condition of originally enabled events, these events should be disabled in a purely asynchronous semantics.

The SBML Level 3 choice¹ is to keep a very flexible semantics, with semi-asynchronous events, which can use either the values at the time they were enabled, or the *current values* at the time they are actually executed, after the execution of the simultaneous events with higher priority, and which can specify the *permanence* of an event in order to define if it should be fired even if its trigger has been invalidated by previous events firing at the same time.

In BIOCHAM, there are no priorities, and the events that are enabled simultaneously are executed in the order of their writing using current values. An event with n assignments of f_i to s_i is therefore equivalent to the sequence of n events with the same trigger for each assignment f_i to s_i . The semantics of events implemented in BIOCHAM can thus be defined in SBML Level 3 using the current value and permanence options and priorities corresponding to the order of writing.

¹The Versions and Releases of the SBML Level 3 Core specification and officially-supported Level 3 package specifications are available at: http://sbml.org/Documents/Specifications/SBML_Level1_3.

2.2. Hybrid continuous-stochastic models



It is worth noting that in SBML Level 3, thanks to the Distributions package, random variables can be represented. This would be useful in the following sections to implement stochastic semantics with SBML events. However, since the SBML Test Suite Database and the page of the Distributions package show that there is apparently no software currently able to cope both with prioritized events and random variables, we have implemented a simple linear pseudo-random number generator (PRNG) using SBML events. The files generated by our hybridization interfaces can therefore be run with any SBML Level 3 core compatible simulator.

2.2 Hybrid continuous-stochastic models

Chemical reactions, originated from random collisions of particles, are discrete and stochastic in nature. Although there is no way to predict the exact state of a chemical system at a specific time point, its *statistical* behavior can be effectively calculated from known probabilistic properties, as done by Gillespie's stochastic simulation algorithm (SSA) [29], to be detailed in Section 2.2.1. The SSA simulation can be especially slow if one or more of the reactions have fast reaction rates (or high event occurrences) because the next reaction time will be very short due to the high probability of firing (one of the) fast reactions.

Despite the fact that all reactions are innately stochastic, those with large reactant counts and high reaction rates can be accurately approximated in terms of deterministic behavior expressed by ODEs. By incorporating both continuous and stochastic semantics into one simulator, an optimal balance between simulation runtime and accuracy can be achieved. This potentially lifts the scalability of simulating large biological systems. In Section 2.2.2, we provide an event-based view

2.2. Hybrid continuous-stochastic models



on the SSA, that serves as basis to a hybrid continuous-stochastic simulator built upon an ODE simulator with events.

2.2.1 Gillespie's stochastic simulation algorithm (SSA)

A reaction model with kinetic expressions can be interpreted under the stochastic semantics as a continuous-time Markov chain (CTMC). A CTMC can be simulated with a stochastic simulation algorithm (SSA), for example, *Gillespie's direct method* [29]. Rather than solving all possible trajectories' probabilities as in the case of Master equations, the algorithm generates statistically correct trajectories.

Gillespie's direct method first calculates *when* the next reaction will occur, then decides *which* reaction should occur with the help of a random number generator. The probability that a certain reaction i will be the next one is determined by the propensities α of the reactions: $\alpha_i = (\text{\#combinations of reactants}) \cdot k_i$ where k_i is the rate coefficient of reaction i . The algorithm repeats the following steps.

1. Calculate *how long from now* (Δt) the next reaction will occur as a Poisson event.

$$\Delta t = \frac{-1}{\sum_j \alpha_j} \cdot \log(\text{ran}_1),$$

where ran_1 is a random number within range $[0, 1]$ and the α_j are propensities at the current state.

2. Choose which reaction will occur according to the probability distribution of reactions. This is done by generating a random number ran_2 within range

2.2. Hybrid continuous-stochastic models



$[0, 1]$, and letting the reaction i be chosen for

$$\frac{\sum_{k=1}^{i-1} \alpha_k}{\sum_j \alpha_j} < \text{ran}_2 \leq \frac{\sum_{k=1}^i \alpha_k}{\sum_j \alpha_j}.$$

3. Update the numbers of molecules to reflect the execution of reaction i , and set current time to $t = t + \Delta t$.

2.2.2 Event-based implementation of SSA

By considering every firing of a chemical reaction as one firing of an event, the *event* semantics of Section 2.1 enables a direct embedding of stochastic reactions into an intrinsically continuous framework without additional implementation of a separate stochastic simulation algorithm. Under this framework, *time* is the only unifying variable to keep track of current state at each instant. This event-based approach permits the simple integration of ODE and stochastic simulation as will be elaborated in Section 2.2.3.

Notice that, in the SSA of Section 2.2.1, *when* the next reaction will occur is independent of *which* reaction will occur, and only one reaction is chosen each time. These facts make it possible for *the complete set of stochastic reaction rules* to be interpreted correctly as *a single event*. Essentially the simulation can be accomplished by compiling the *when* and *which* questions Gillespie's direct method asks into an event. Specifically the event is triggered by the calculated next reaction time (τ); the event obtains a new random variable (ran) and then conditionally updates the particle counts depending on which reaction is chosen to occur next. To accommodate all stochastic rules in one event, each update entry is composed of conditional expressions over the propensities and the random number that decides

2.2. Hybrid continuous-stochastic models



which reaction occurs.

Example 2.1 Given the stochastic reaction rules $A + 2B \xrightarrow{k_1} C$ and $C \xrightarrow{k_2} 2A$ from [29], we derive their propensities by

$$\text{alpha1} = k_1 \times (nA) \times \frac{(nB) \times (nB - 1)}{2}$$

$$\text{alpha2} = k_2 \times (nC)$$

where “ nX ” denotes the particle count of species X . Then the next reaction time from the current time point can be decided by

$$e = \frac{-1}{\text{alpha_sum}} \cdot \log(\text{ran}_1)$$

for ran_1 a random number within $[0, 1]$ and where $\text{alpha_sum} = \text{alpha1} + \text{alpha2}$. The first reaction is chosen for the next occurring reaction if $0 < (\text{alpha_sum} \times \text{ran}_2) \leq \text{alpha1}$, which leads to the consumption of one A and two B 's and producing one C .

This is achieved by the following event:

```
1 event (Time>tau ,
2       [ran , tau , ran , nA , nB , nC] ,
3       [rand , Time + e , rand ,
4         if alpha_sum*ran=<alpha1 then nA-1 else nA+2 ,
5         if alpha_sum*ran=<alpha1 then nB-2 else nB ,
6         if alpha_sum*ran=<alpha1 then nC+1 else nC-1]) .
7 macro (rand , seed) .
```

Where `rand` is a macro implementing a PRNG as explained in Section 2.1.2 and `e` is defined as shown above. Note that the update of the particle counts of the first reaction is reflected in the three `then` entries, and that of the second reaction is reflected in the three `else` entries. Note also that a single parameter `ran` is used

2.2. Hybrid continuous-stochastic models



twice to store first ran_1 then ran_2 , this is in order to use our simple linear PRNG with a single seed.

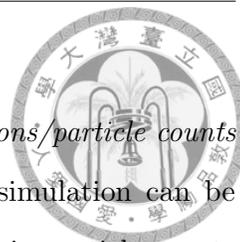
This encoding relies on the left to right ordering of the different events associated to a single trigger (see Section 2.1.2). This ordering is imposed to two kinds of parameters – the random number, and a reaction’s propensity function – such that possible errors are avoided. Because the two kinds of parameters depend on the *current* number of molecules, they are listed in front of molecular species. So their values are not changed before the *completion* of reaction firing, that is, all species’ counts have been updated according to the chosen reaction.

2.2.3 Preprocessor for composing continuous and stochastic models

The purpose of our *preprocessor* for composing heterogeneous biochemical models automatically is to provide a simple interface for specifying hybrid simulations without digging into algorithmic details. The only work for the user is to decide the semantic model for each of the reactions under simulation. The models are then *processed* into a composed hybrid model suitable for simulation or analysis.

In classical work on hybrid simulation [4, 51], chemical reactions are divided into two groups according to their *propensities* and *reactants’ concentrations*: one consisting of reactions to be simulated stochastically using SSAs, and the other consisting of reactions to be simulated deterministically using ODEs. The former is referred to as the *stochastic reactions* and the latter *continuous reactions*. While continuous reactions simply advance continuously according to their governing ODEs with the pass of time, stochastic reactions fire discretely in time with frequency

2.2. Hybrid continuous-stochastic models



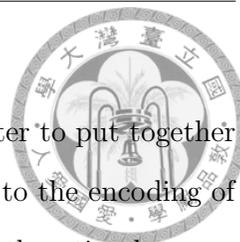
determined by their propensities. When the *reactant concentrations/particle counts* and the *propensity* of a reaction are sufficiently large, ODE simulation can be faithfully applied without introducing unacceptably large errors in particle counts when using the total counts of corresponding species as references. (i.e. keeping the ratio $\frac{|nX_{\text{expected}} - nX_{\text{ODE}}|}{nX_{\text{ODE}}}$ acceptably small, where nX_{expected} is the expected particle count of species X in fully stochastic simulations, and nX_{ODE} is the particle count obtained when ODE simulation is allowed.) At the same time, frequent updates of particle counts within a small time interval are avoided, thus accelerating the simulation speed.

Hybrid species are referred to as those involved in both stochastic and continuous reactions. This kind of species requires special attention because they are influenced by two different mechanisms: *ODEs* that govern differential behavior by continuously changing related *concentrations*, and *events* that regulate stochastic behavior by modifying *particle counts* discretely whenever triggered. So a hybrid species is under two kinds of modification: one targets at the evolution of macroscopic concentrations and the other targets at the changes in microscopic particle counts.

In our implementation, a fresh new variable is introduced for each hybrid species to represent its total quantity (the summation of the numbers of particles from both continuous and stochastic models). That variable is set equal to the sum of a continuous variable multiplied by the corresponding volume and a small discrete number of particles. ODEs will act on the continuous part, whereas discrete events will impact the discrete one². In all kinetic expressions (i.e. in rate equations and propensity functions), the hybrid species are expressed by the corresponding new

²This specific implementation is related to the constraint that, contrary to the SBML specification, BIOCHAM continuous variables cannot be modified by events.

2.2. Hybrid continuous-stochastic models



variables representing the total amount. It is then a simple matter to put together the ODEs for the continuous part and the events corresponding to the encoding of the stochastic part as described in the previous section. It is worth noting however that while the total amount of each species is guaranteed to be nonnegative, the continuous part can sometimes become negative.

Example 2.2 The reaction model of bacteriophage T7 infection described in [4] is an interesting example that can be hybridized by static partitioning of the reactions with continuous semantics for protein synthesis and with stochastic semantics for gene activation. The partition in [4] consists in taking the fifth and sixth reactions with the continuous semantics and the other reactions with stochastic semantics, as follows:

```
1 % Continuous reaction rules      1 % Stochastic reaction rules
2 MA(c5) for tem => tem+struc.     2 MA(c1) for gen => tem.
3 MA(c6) for struc => _ .         3 MA(c2) for tem => _ .
4                                     4 MA(c3) for tem => tem+gen.
5 parameter(c5, 1000).           5 MA(c4) for gen+struc => virus.
6 parameter(c6, 1.99).           6
7 parameter(c1, 0.025).          7 parameter(c1, 0.025).
8 parameter(c2, 0.25).          8 parameter(c2, 0.25).
9 parameter(c3, 1).             9 parameter(c3, 1).
10 parameter(c4, 0.0000075).    10 parameter(c4, 0.0000075).
```

In this example, `tem` and `struc` are hybrid species representing respectively the template viral nucleic acids and the viral structural proteins, while `gen` and `virus` are purely stochastic and represent the genomic viral nucleic acids and the final virus. The full input files with parameters and output file after preprocessing in both BIOCHAM and SBML are available at http://lifeware.inria.fr/supplementary_material/TOMACS/Alfonsi/. All experiments in this chapter are conducted in BIOCHAM on a 2.9GHz Intel Core i7 platform with 16GB 1600MHz DDR3 memory.

2.3. Dynamic strategies for hybrid continuous-stochastic simulations



Table 2.1 summarizes the result from 1000 simulations over a time horizon of 100 days. The experimental results shows that the hybrid simulation improves by three orders of magnitude the simulation time. The accuracy of the hybrid simulation technique will be demonstrated in more detail in Example 2.5.

method	#fired events	CPU time (sec)
stochastic	276556	218.7
hybrid	832	0.75
ratio	0.003	0.003

Table 2.1: A comparison between purely stochastic and hybrid simulation implemented using chemical reactions and events. Columns *#fired events* and *CPU time* respectively hold the number of events triggered and runtime in seconds. All values are the average of 1000 simulations over a time horizon of 100 days. The last row shows the ratio of hybrid to stochastic statistics.

2.3 Dynamic strategies for hybrid continuous-stochastic simulations

The above discussion assumes a static partition of a set of reactions into two subsets interpreted under continuous and stochastic semantics. Once the partition is established based on the system's initial conditions and partition criteria, it stays *fixed* throughout a simulation process. However, such a partition strategy may be inadequate for two reasons: firstly, a good static partition may not be known *a priori* given only initial conditions, secondly, a good static partition may not even exist. Essentially a fixed semantic interpretation of a reaction can lead to inaccurate and/or inefficient simulation when the reaction's reactants' counts and/or its propensity fluctuate substantially over time, thus violating the legitimacy of abstraction with continuous semantics and/or being unnecessarily trapped in the too frequent firing of reaction events. It is therefore desirable to adjust the reaction partition *dynamically*

2.3. Dynamic strategies for hybrid continuous-stochastic simulations



along the progress of a simulation.

2.3.1 Dynamic partitioning criteria

Particle count and *propensity value* [4] are predominant factors of choice between the stochastic and continuous interpretation of reactions. Examples of other higher-level factors derived from the two include: critical relative fluctuation [8] that describes a reaction's influence on a species' count relative to each one's total count; particle count of substrate involved, and ratio of a reaction's propensity to the sum of all reactions' propensities [62]. In [75] however, the partitioning criteria themselves, composed of particle count and propensity value, do not possess explicit meanings, rather they are derived to guarantee that the error of each approximation is smaller than the user-specified value.

We adopt a partition strategy that takes both particle counts and propensities into account: A reaction can be interpreted as differential only if its propensity value exceeds some target threshold *and* its related species' particle counts all exceed a certain threshold. In the sequel, we will refer to the two threshold values as *propensity threshold* and *particle count threshold*, respectively. To preserve flexibility on the user's side to decide the trade-off between accuracy and efficiency, both non-negative thresholds can be tuned by users according to the need. Increasing the value(s) leads to more accurate and less efficient simulations, while lowering the value(s) leads to more efficient but less accurate simulations. Note that a threshold's value can be set to zero if the accuracy degradation caused by its corresponding property is assumed to be non-substantial.

Consider the SBML reactions of Section 2.1.1. For reaction i with $\sum_j l_{ij} \times S_j \rightarrow$

2.3. Dynamic strategies for hybrid continuous-stochastic simulations



$\sum_j r_{ij} \times S_j$, under the time step size Δ of the ODE simulator used, by default we let

$$\text{propensity threshold} = (n_1 \times \Delta)^{-1}, \text{ and}$$

$$\text{particle count threshold} = n_2 \times \max_{i,j} |r_{ij} - l_{ij}|,$$

where n_1 and n_2 are two parameters of non-negative real values. To determine the value of n_1 , note that the expected time period from present to a reaction's next firing equals the reciprocal of its propensity value. To avoid simulation being trapped by the frequent firing of a fast (with respect to Δ) reaction, we can interpret a reaction as continuous only if its expected time period from present to its next firing is shorter than the reciprocal of the propensity threshold. Thereby we may potentially skip unnecessarily many event updates. The smaller the value of n_1 is, the lesser the efficiency is gained from continuous semantics. On the other hand, to determine n_2 , note that $\max_{i,j} |r_{ij} - l_{ij}|$ is the largest possible change in particle count by one reaction firing among all reactions. For continuous semantics to be legitimate, particle counts should be large enough. Furthermore, for continuous semantics to be a good approximation, the change in the particle count of each species by one reaction occurrence should be relatively small compared to the species' total count. The larger the value of n_2 is, the more stringent the condition is for a reaction to be interpreted as continuous.

2.3.2 Implementation

There are two directions of semantic switching in dynamic partitioning: (1) from continuous to stochastic, and (2) from stochastic to continuous. During simulation, instead of monitoring the switching criteria all the time, the reactions are checked

2.3. Dynamic strategies for hybrid continuous-stochastic simulations



against the criteria within the event that realizes stochastic reactions, and only at the time of reaction firing. At the start of a simulation, all reactions are classified as stochastic by default. When a reaction event is triggered, apart from updating the particle counts according to the reaction selected, all reactions are checked against the user-specified criteria whether they are eligible for continuous semantics. The eligibility is usually based on the requirement of being theoretically sound (for accuracy concern) and can favorably include being practically beneficial (to improve efficiency). Switching from continuous to stochastic occurs when a reaction no longer satisfies the criteria; a reaction is switched from stochastic to continuous if its current condition can satisfy the criteria.

For the first switching direction, postponement in switching can result in accuracy degradation. Indeed, one continuous reaction requires switching to stochastic only when the small error assumption for continuous interpretation is no longer satisfied. With our event formulation, the delay is *at most* the time period between now and the next reaction time of current set of stochastic reactions, provided that there is at least one stochastic reaction. When there is no stochastic reaction, the sum of propensities will be zero and thus resulting in infinite waiting time till the next reaction. To avoid this infinite waiting problem, the absence of stochastic reactions can be detected to enforce progress in simulation (this is achieved by the last macro, i.e., function definition, as shown in the BIOCHAM code of Example 2.3).

For the second switching direction, postponement in switching does not lead to loss of accuracy, although early switching can improve simulation efficiency. Since switching in both directions are realized in the same event, the upper bound of the delay is the same as that of the first switching direction. To make the most out of the unavoidable trade-off between accuracy and efficiency, once the partitioning strategy

2.3. Dynamic strategies for hybrid continuous-stochastic simulations



and its corresponding criteria are set, the goal becomes one of always maximizing the set of continuous reactions without violating the criteria.

2.3.3 Simple example

The following example shows the general implementation of dynamic partitioning by SBML events for Example 2.1. Once the partitioning strategy is chosen, the corresponding part of partitioning criteria that determine when a reaction can switch from stochastic to continuous are incorporated into the event as conditions, in ways demonstrated in the example.

Example 2.3 Consider again the system of two reactions: $A + 2B \xrightarrow{k_1} C$ and $C \xrightarrow{k_2} 2A$. The main structure of BIOCHAM code used to fulfill simulation with dynamic partitioning is as follows:

```
1 % Continuous semantics
2 MA(k1_diff) for A + 2*B => C.
3 MA(k2_diff) for C => 2*A.
4
5 % Event for stochastic semantics and dynamic partitioning
6 event(Time>tau,
7     [ran, tau, ran, k1_diff, k1_stoch, k2_diff, k2_stoch,
8     nA, nB, nC],
9     [rand, Time + e, rand,
10     if (condition for reaction 1 to be continuous is
11     satisfied)
12     then k1 else 0,
13     if k1_diff=0 then k1 else 0,
14     if (condition for reaction 2 to be continuous is
15     satisfied)
16     then k2 else 0,
17     if k2_diff=0 then k2 else 0,
18     if alpha_sum*ran =<= alpha1 then nA-1 else nA+2,
19     if alpha_sum*ran =<= alpha1 then nB-2 else nB,
20     if alpha_sum*ran =<= alpha1 then nC+1 else nC-1]).
21
22 % Hybrid species
```

2.3. Dynamic strategies for hybrid continuous-stochastic simulations

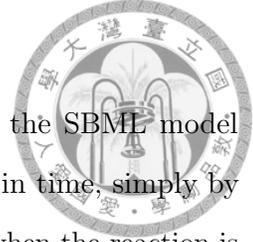


```
20 macro(A_total, [A]*volume + nA).
21 macro(B_total, [B]*volume + nB).
22 macro(C_total, [C]*volume + nC).
23 macro(alpha1, k1*A_total*B_total*(B_total-1)/2).
24 macro(alpha2, k2*C_total).
25 macro(alpha_sum, alpha1 + alpha2).
26 macro(e, if alpha_sum=0
27         then (-1/propensity threshold)
28         else (-1/alpha_sum)*log(ran)).
```

Under dynamic partitioning, all species are treated as hybrid continuous-stochastic species; each reaction can become either continuous or stochastic, but not both, at any time point. Specifically, each rate constant k_i is duplicated into k_{i_diff} and k_{i_stoch} , to simplify the process of semantic switching to value alteration between 0 and the real value of rate constant. A reaction r is *stochastic* if and only if k_{r_diff} is set to 0 and k_{r_stoch} is set to the r 's rate constant value, while it is *continuous* if and only if k_{r_stoch} is set to zero and k_{r_diff} is set to the value of its natural rate constant.

The last macro decides the next reaction time of the set of stochastic reactions, which is also the next time point for checking and adjusting the partition until consistent with the strategy imposed. The *else* part is the same as that in the static partitioning, which implements Gillespie's Direct Method as described previously in Section 2.2.1. The *then* part serves to avoid, when all reactions become continuous, the problem of infinite waiting time before the next reaction. Note that the value is also the upper bound of semantic switching delay, which is set here to be the average firing period of the fastest possible stochastic reaction under current strategy. This is to make sure that the average particle count error resulted from delayed switching to stochastic semantics will not exceed the species' stoichiometric number in the reaction.

2.3. Dynamic strategies for hybrid continuous-stochastic simulations

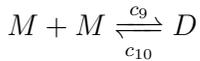
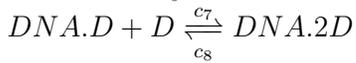
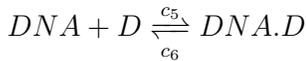
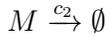


Note that this encoding allows to trace which reactions of the SBME model were chosen to be stochastic (resp. continuous) at which point in time, simply by observing the value of $k_{i-stoch}$ (resp. k_{i-diff}), which is non-null when the reaction is stochastically (resp. continuously) evaluated.

2.3.4 Performance evaluation

The effectiveness of dynamic over static partitioning by our proposed framework is evaluated in Examples 2.4 and 2.5 below. Additionally, implementations of different partitioning strategies and a comparison among them is presented in Example 2.5.

Example 2.4 We study Goutsias model [40] to demonstrate the effectiveness of dynamic partitioning. The model describes the transcription regulation of a repressor protein M in bacteriophage λ . It involves 6 different species and 10 reactions listed as follows:



Assume the particle counts and parameters are initialized as follows:

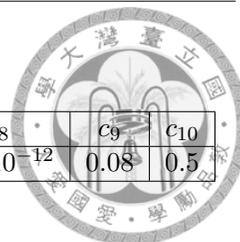
$$\#RNA_{t=0} = \#DNA.D_{t=0} = \#DNA.2D_{t=0} = 0$$

$$\#M_{t=0} = \#D_{t=0} = 10$$

$$\#DNA_{t=0} = 2$$

2.3. Dynamic strategies for hybrid continuous-stochastic simulations

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
0.043	7×10^{-4}	71.5	3.9×10^{-6}	0.02	0.48	4×10^{-4}	9×10^{-12}	0.08	0.5



For static partition used in this example, reactions $M + M \xrightleftharpoons[c_{10}]{c_9} D$ are interpreted under differential semantics while all other reactions are stochastic. The partition is based on the fact that molecules M and D have the greatest initial counts, and both have initial propensities no less than 5 while all other reactions' initial propensities are much smaller than 1. As for dynamic partition used, the propensity threshold and the particle count threshold are set to 5 (with $n_1 = 20$) and 20 (with $n_2 = 10$), respectively; a reaction is interpreted as continuous only if its propensity value exceeds the propensity threshold *and* its related species' particle counts all exceed the particle count threshold. This criterion aims to take both population and propensity into account for the following reasons: firstly, in this model, discreteness from extremely low particle counts is the main cause of violation to the continuous semantics' assumption. Secondly, the rate constants of the system span orders of magnitudes, even among reactions with shared reactants. So it can be highly probable that the large difference in reaction rates can introduce inefficiency during simulation.

With both static and dynamic strategies partitioning reactions into continuous and stochastic based on the same considerations, i.e. particle counts *and* propensities, the only major difference between the two strategies is the allowed time point for information gathering and making corresponding semantic alterations. For static strategy, reactions are partitioned once and for all based on initial particle counts and propensities. Dynamic strategy, on the other hand, updates the partition according to current system state whenever an event is triggered. Figure 2.1 shows the average results from 1000 simulations. Note that even as static partition strategy has taken initial conditions into account, the difference between static partition strategy and

2.3. Dynamic strategies for hybrid continuous-stochastic simulations



the expected result (obtained by averaging over 1000 fully stochastic simulations) is already much larger than that of dynamic partition strategy after 5 time units.

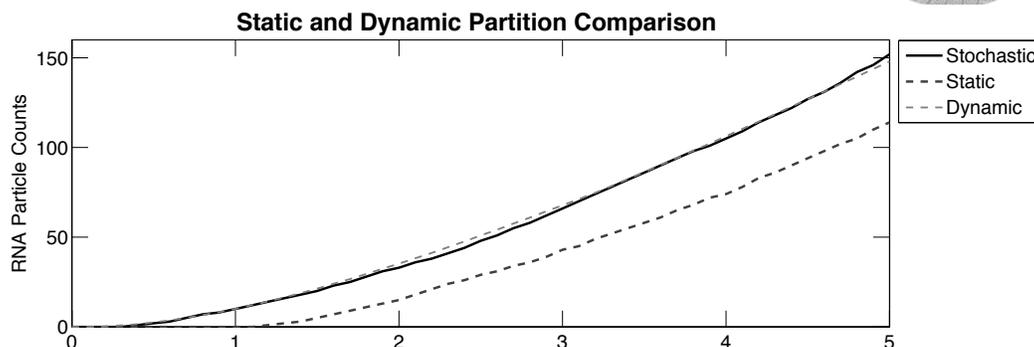


Figure 2.1: Comparison of static and dynamic partition strategy with stochastic simulation result. Each curve represent the average of 1000 simulation runs of corresponding strategy, with simulation horizon = 5 time units.

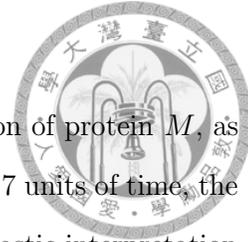
Apart from the accuracy improvement shown in Figure 2.1, a substantial reduction on the firing of events and thus CPU time is achieved by the dynamic partition, as is shown in the last two rows of Table 2.2. Notice that the reduction on event count is more substantial than the reduction on run time because of the extra checking needed in the dynamic partition to decide potential switchings at each event firing.

method	#fired events	CPU time (sec)
purely stochastic	141 036.91	96.07
static partition	9931.68	9.67
dynamic partition	126.42	1.61
ratio over stochastic	0.000 896	0.0168
ratio over static partition	0.0127	0.166

Table 2.2: Average number of events fired and average runtime from 100 simulations with simulation horizon set to 100 time units, comparing over three simulation methods. The last two rows are the ratios of dynamic partition strategy's statistics to that of purely stochastic and static partition strategy's, respectively.

Figure 2.2 explains these results by showing the behavior of the dynamic partitioning strategy in this example. On the long time horizon, the dynamic strategy interprets

2.3. Dynamic strategies for hybrid continuous-stochastic simulations



reactions $\{1, 9, 10\}$, i.e., the production and reversible dimerization of protein M , as continuous and the other ones as stochastic. However, on the first 7 units of time, the dynamic strategy applies a completely different choice, with stochastic interpretation for those reactions and reaction 3, the RNA production, continuous. Then, for a transient time of around 20 units, reactions $\{1, 3, 9, 10\}$ are mainly continuous with a decreasing frequency for reaction 3.

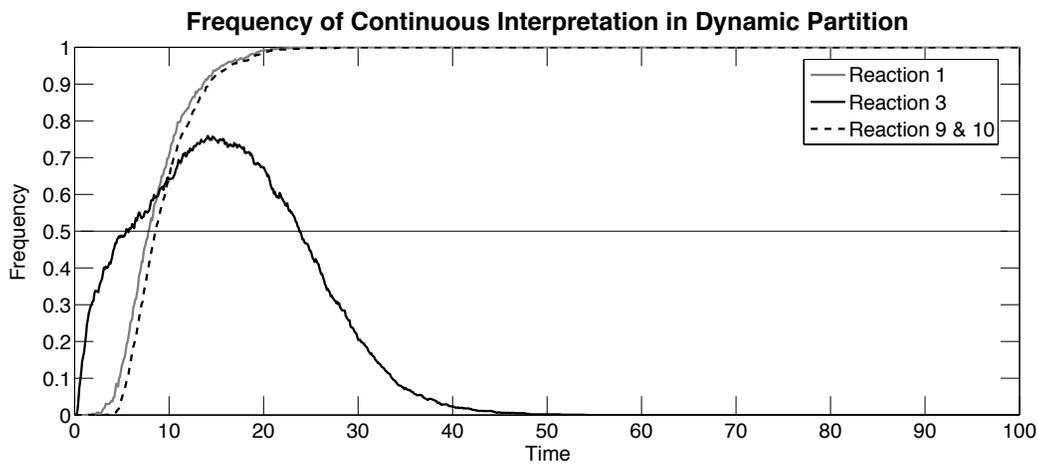


Figure 2.2: The frequency of each reaction being interpreted as continuous under dynamic partition strategy, over time horizon of 100 units, and calculated over 1000 simulations. Reactions not listed are never interpreted as deterministic during the simulation horizon.

Example 2.5 Let us consider again the model of intracellular growth of bacteriophage T7 of Example 2.2 with the static partitioning strategy of [4], noted $\{1, 2, 3, 4\}$ since the first four reactions are always stochastic and the last two ones always continuous, and with a different static partition $\{1, 3\}$ in which only the first and third reactions are stochastic, the others being continuous. For dynamic partition, the propensity threshold and the particle count threshold are set to be 10 (with $n_1 = 10$) and 5 (with $n_2 = 5$), respectively.

2.3. Dynamic strategies for hybrid continuous-stochastic simulations

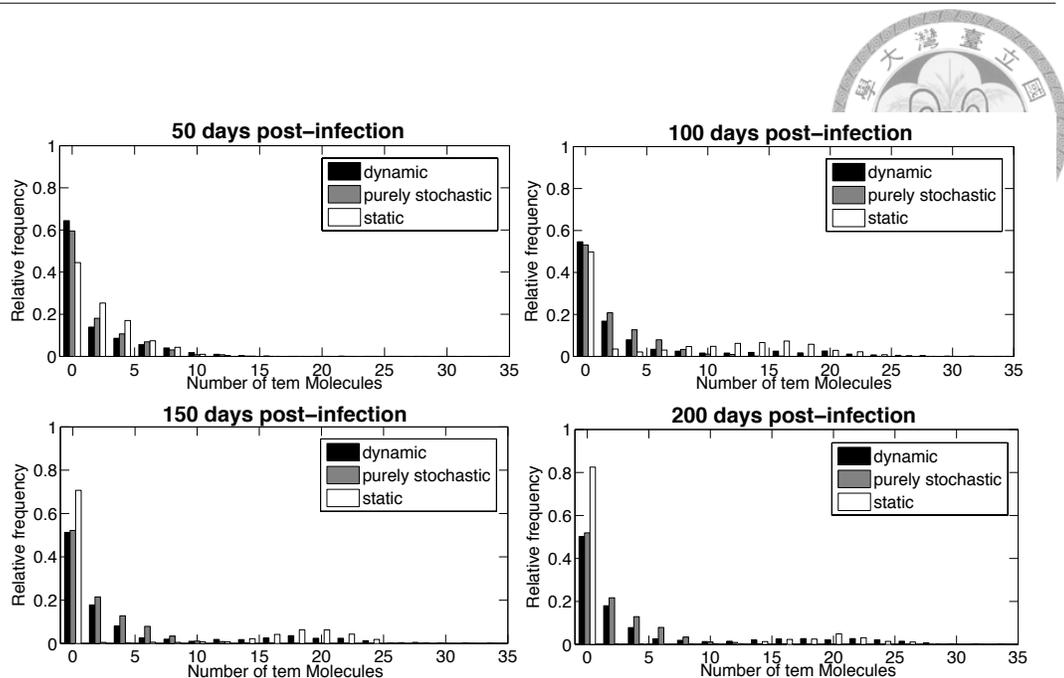


Figure 2.3: Comparison of post-infection distributions of *tem* particle counts obtained at time= 50, 100, 150, 200 days, by stochastic, static hybrid and dynamic hybrid simulations (based on 1000 simulation runs of each strategy.)

Figure 2.3 depicts the relative frequencies of the numbers of *tem* molecules after 50, 100, 150, 200 days, obtained with that static partition, with the dynamic partitioning strategy, and with SSA. Each bar represents the relative frequency of *tem* molecule count falling in that region after certain amount of time. As can be clearly seen in the graph, bars of static partition deviate from those of purely stochastic simulation, while bars of dynamic partition are closer to the purely stochastic ones.

These observations can be made quantitative using statistical distances. Let us use the *two sample Kolmogorov-Smirnov test* as distance measure (KS distance) to compare the relative frequencies. Table 2.3 shows the KS distance between the distributions obtained by SSA and the static partitioning $\{1, 2, 3, 4\}$ of [4], the static partitioning $\{1, 3\}$ and the dynamic partitioning respectively.

2.4. Hybrid Boolean models



Post-infection Time (days)	50	100	150	200
KS distance SSA - static hybrid {1, 2, 3, 4}	0.0525	0.7145	0.8035	0.836
KS distance SSA - static hybrid {1, 3}	0.3815	0.9225	0.624	0.6055
KS distance SSA - dynamic hybrid	0.0515	0.116	0.1485	0.161

Table 2.3: Post-infection distributions of *tem* molecules from simulations using different hybrid strategies compared to the reference fully stochastic model. Each row contains the outcome of applying two-sample Kolmogorov-Smirnov test on the distributions obtained from 1000 simulations using specified hybrid strategy and the reference fully stochastic model. The smaller the value, the more similar the two distributions involved. Distributions at four sampling points are used for comparison through the evolution of time, as listed in four corresponding columns.

By taking the particle count distribution of purely stochastic simulation as the reference, this example shows that the dynamic strategy always beats the static partition strategies and improves the accuracy of the simulation to a small distance from SSA along all time points.

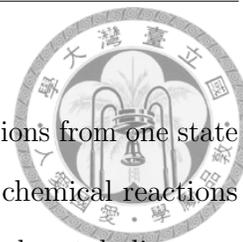
2.4 Hybrid Boolean models

In this section we demonstrate how Boolean models can also be composed with continuous and even hybrid continuous-stochastic models in SBML.

2.4.1 Preprocessor for composing continuous and Boolean models

In this section, we consider the composition of continuous reaction models with Boolean transition systems. One typical use of this form of composition is for modeling the interactions between gene expression and metabolism on different time scales. Gene networks can be modeled by simple Boolean regulatory networks

2.4. Hybrid Boolean models



representing the on/off states of the genes and the possible transitions from one state to another, while metabolic networks are naturally modeled by chemical reactions with continuous semantics. Hybrid models of gene expression and metabolism can thus be naturally built as hybrid continuous-Boolean models, and analyzed and simulated as such.

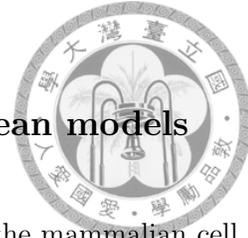
A continuous-Boolean composition necessitates specifying:

- the link between the discrete/continuous variables and the Boolean variables, e.g. by fixing particle count or concentration threshold values,
- the relationship between the discrete logical time of the Boolean model and the continuous real time of the continuous reaction model, e.g. by adding delays on Boolean transitions,
- the integrity constraints between both dynamics.

There is currently no general method for these tasks. Our high-level interface takes as input

1. a reaction model that accommodates both stochastic and continuous semantics,
2. a Boolean transition system,
3. an interface specifying for each Boolean transition, the triggers and actions on the reaction model variables,

and produces as output a system of reactions and events which synchronize the execution of both input models.



2.4.2 Hybrid composition of continuous-Boolean models

In [71], Singhanian et al. have proposed a simple hybrid model of the mammalian cell cycle regulation. This cell cycle model of low dimension has been evaluated in terms of flow cytometry measurements of cyclin proteins in asynchronous populations of human cell lines. The few kinetic constants in the model are easier to estimate from the experimental data than the numerous kinetic constants of a single large ODE model.

In this model, cyclin abundances are tracked by piecewise linear continuous equations for cyclin synthesis and degradation. Cyclin synthesis is regulated by transcription factors whose activities are represented by discrete variables (0 or 1) and likewise for the activities of the ubiquitin-ligating enzyme complexes that govern cyclin degradation. The discrete variables change according to a predetermined sequence, with the times between transitions determined by the amount of cyclin presented as well as exponentially distributed random variables.

This model can be reconstructed using our interface as the hybrid composition of a purely continuous reaction model of cyclin activation and degradation, with a Boolean model of cell cycle phase transitions. We provide here the real examples and thus the ASCII syntax for the BIOCHAM constructs described in Section 2.1.1. Beside the syntax introduced before, the `present` command specifies the initial concentration, and the `macro` command defines a function that makes the reaction rates dependent on the value of boolean variables, as specified in the original article.

The inputs are:

1. *the continuous reaction model of cyclin activation*, which provides an always

2.4. Hybrid Boolean models



progressing continuous behavior:

```
1 % Initial Conditions
2 present(CycA, 1).
3 present(CycB, 1).
4 present(CycE, 1).
5
6 % Reaction Rules
7 k_sa for _ => CycA.
8 MA(k_da) for CycA => _.
9
10 k_sb for _ => CycB.
11 MA(k_db) for CycB => _.
12
13 k_se for _ => CycE.
14 MA(k_de) for CycE => _.
15
16 macro(k_sa, 5+6*B_tfe+20*B_tfb).
17 macro(k_sb, 2.5+6*B_tfb).
18 macro(k_se, 0.02+2*B_tfe).
19 macro(k_da, 0.2+1.2*B_cdc20a+1.2*B_cdh1).
20 macro(k_db, 0.2+1.2*B_cdc20b+0.3*B_cdh1).
21 macro(k_de, 0.02+0.5*B_scf).
```

2. *the Boolean transition system of the cell cycle progression*, which is given in [71] as the following limit cycle of state transitions. The `add_boolean_state` command defines a numbered state, and associates the boolean variables true in that state; the `add_boolean_transition` command defines a named transition between two states. Here is an excerpt of the file:

```
1 % States and corresponding active boolean species
2 add_boolean_state(1, [B_cdh1]).
3 add_boolean_state(2, [B_tfe, B_cdh1]).
4 add_boolean_state(3, [B_tfe]).
...
1 set_initial_boolean_state(1).
2
3 % Transitions between states
4 add_boolean_transition(T12, 1, 2).
```

2.4. Hybrid Boolean models



```
5 add_boolean_transition(T23, 2, 3).
```

...

3. *the synchronization between both models*, specified as a set of triggers and actions (similar to the ones in events described in Section 2.1.2) associated to the Boolean transitions via the `add_boolean_transition` command. In this hybrid model, the time for the Boolean transitions are given by random variables. This is represented by a parameter `tau` and a macro `next_event` as can be seen in the following excerpt:

```
1 parameter(tau, 0).
2 macro(next_event, Time - lambda * log(ran)).
3 event(Time = 0, [ran, tau], [rand, next_event]).
4
5 parameter(theta_e, 80).
6 parameter(theta_a, 12.5).
7 parameter(theta_1_b, 21.25).
8 parameter(theta_2_b, 3).
9
10 add_interface(T12, Time > tau, [ran, lambda, tau], [rand,
    0, next_event]).
11 add_interface(T23, Time > tau and [CycE] * masst >=
    theta_e,
12 [ran, lambda, tau], [rand, 0.01, next_event]).
```

...

The result of the composition is an SBML model formed of the continuous reaction model augmented with a list of events. The events implement the Boolean transition cycle from state 1 to 9 and back to 1, and their synchronization with the continuous reaction model. In this form, the hybrid model can be simulated using any simulator of SBML models. The simulation over a time horizon of *100 hours* takes 150 ms. The simulation result is shown in the upper plot in Figure 2.4.

2.4. Hybrid Boolean models

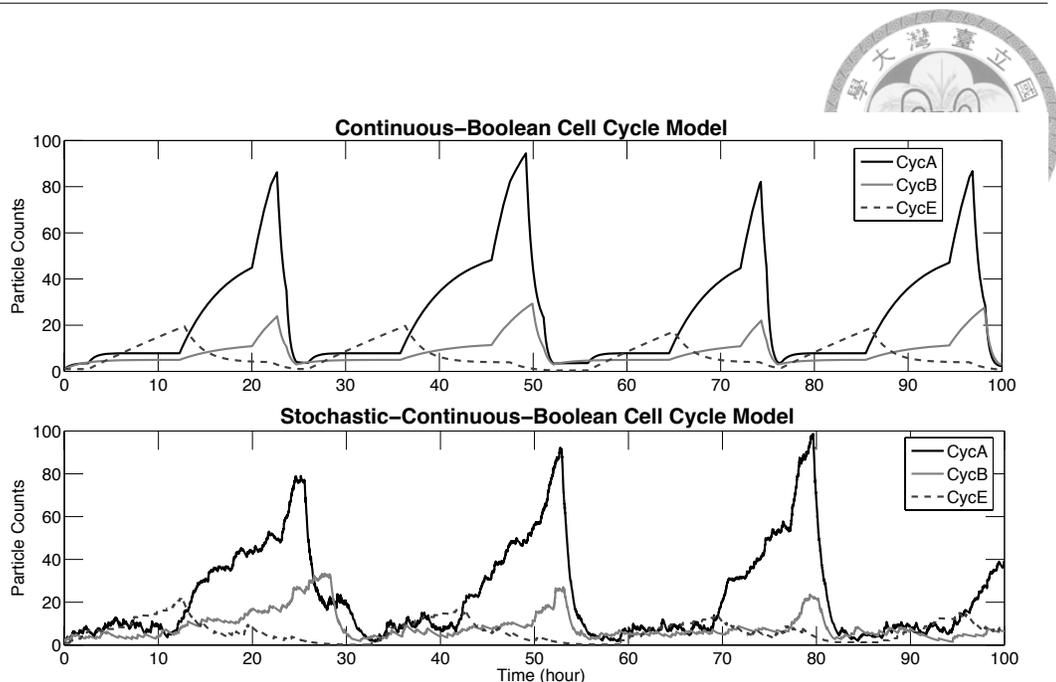


Figure 2.4: Evolution of the cyclin’s particle counts in Singhanian et al. model of the cell cycle. Simulation horizon = 100 hours. (Upper) original continuous-Boolean model (with stochastic delays), with average runtime = 0.15 seconds; (Lower) hybrid stochastic-continuous-Boolean model, with average runtime = 8.42 seconds.

2.4.3 Stochastic-continuous-Boolean model simulation

A continuous-Boolean model can be easily generalized to a more realistic stochastic-continuous-Boolean one by extending the purely continuous reaction model to a stochastic-continuous one, using event-based methods as shown in Section 2.2.2 and Section 2.2.3.

In the lower plot in Figure 2.4, we demonstrate the simulation result of a stochastic-continuous-Boolean cell cycle model. This model extends the purely-continuous reaction model of cyclin activation proposed by Singhanian et al., making it more realistic by allowing stochastic semantics for reactions. In the model simulated here, cyclin synthesis reactions are stochastic and cyclin degradation reactions are continuous.



Chapter 3

Reconfigurable biochemical reaction systems synthesis

In this chapter, we demonstrate our synthesis method that generates from behavior specifications their realizing module-based reconfigurable systems with kinetics and species dependencies described in CRN. The desired behavior can be specified in three formats proposed: whether explicitly as Boolean formula, neuromorphic algorithms based on binary neurons, or implicitly given as data set for the system to learn from. CRNs synthesized in case studies are verified by simulations using *Biocham*.

3.1 Modeling language

Biomolecular system engineering distinguishes itself from the many other engineering efforts with its dependency on underlying system's lower-level properties, such as the interactions between *certain* species, the *adaptive* sets of activated/inactivated

3.2. Reconfigurable logic circuit synthesis



reactions depending on environmental conditions, and the *stochastic* nature of kinetics that can suddenly become dominant due to changes in concentration level or spatial distribution. As a result, even the “same” modules identified from a higher behavior level might not be appropriately modeled with directly duplicated description.

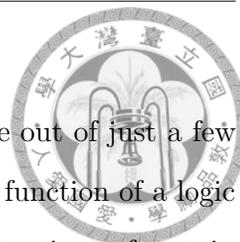
The modeling languages suitable for our design purpose should allow simplified description for more general application, only contain what is necessary, while still able to provide faithful prediction of the behavior realized biochemically. Consequently during simplification, each level of abstraction must be performed with care not to obscure seemingly unimportant information that can actually end up crucial through the levels. One example is the applicability of approximating *chemical master equations* by *mass-action kinetics*, the variations neglected might be minor for one reaction, but can easily be amplified after reaction cascades, and spread widely through the the generally tightly connected biological network.

Chemical reaction network (CRN) is chosen as our modeling language. It is able to describe interactions and kinetics that different abstraction levels aim to capture. Besides, CRN can accommodate different semantics interpretations in one unified format, preserving the flexibility to choose the most appropriate one till when the implementing method is decided, reducing the deviation between expected and resulted behavior.

3.2 Reconfigurable logic circuit synthesis

In this section, we propose an FPGA-like reconfigurable system, comprised of two kinds of repetitive modules—configurable logic units and interconnects, both of which are built from biochemical reactions. Our construction is advantageous in the

3.2. Reconfigurable logic circuit synthesis



following three respects. First, a configurable logic unit is made out of just a few reactions and species based on analog computation. Second, the function of a logic unit can be easily configured in realtime by altering the concentrations of certain biochemical species—similar to how organisms adapt their inner functions according to environmental signals received. Third, our construction maintains modularity and composability. The *retroactivity* [21] issue is overcome in the system, that is, composing a system with an extra module cannot invalidate the system's behavior.

Our reconfigurable circuitry consists of two kinds of components: configurable logic units (Sec. 3.2.1) and configurable interconnects (Sec. 3.2.2). Each logic unit (similar to those in silicon FPGAs) has k input ports (each represented by its assigned species) and one or multiple output ports. It can realize a certain set of logic functions up to k inputs. (In our discussion we set $k = 2$ and let the realizable functions be AND, OR, XOR, and NOT.) The logic units can be composed through configurable interconnects.

In this section, we focus on *combinational* logic circuits, where the outputs are functions only of the inputs. Each signal line, after some delay, stabilizes to the value calculated by substituting current input values into the logic function defined by the gates preceding it. This tendency to converge and to stay at steady state ensures that once the signals in a combinational circuit reach the stable values (corresponding to current set of inputs), the values on all signal lines will remain unchanged given that the input values remain constant. We use chemical *equilibrium* to realize the idea of *signal stability* in our FPGA unit design. As a result, the *timing* for *output readout* becomes less critical since the time needed for the designed reactions to reach equilibrium is highly predictable, and the output signals would remain over time.

3.2. Reconfigurable logic circuit synthesis

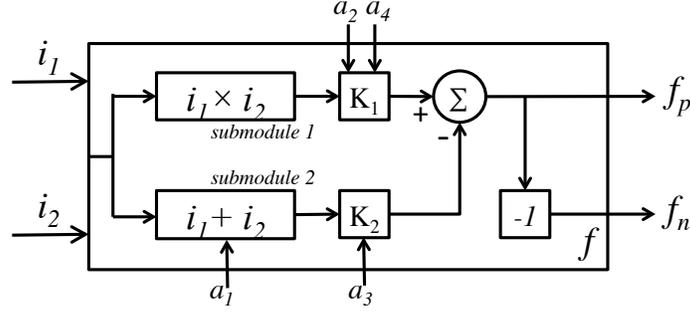


Figure 3.1: Block diagram of configurable logic unit.

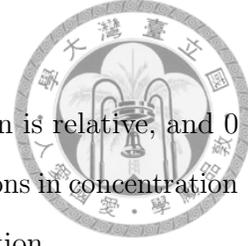
3.2.1 Configurable logic units

The configurable logic unit that we propose is realized based on the following equations of *arithmetic over reals*.

$$\left\{ \begin{array}{l} \text{AND}(i_1, i_2) = -(0 \times (i_1 + i_2) - 1 \times (i_1 \times i_2)) \quad (1) \\ \text{OR}(i_1, i_2) = +(1 \times (i_1 + i_2) - 1 \times (i_1 \times i_2)) \quad (2) \\ \text{XOR}(i_1, i_2) = +(1 \times (i_1 + i_2) - 2 \times (i_1 \times i_2)) \quad (3) \\ \text{NOT}(i_1) = \text{XOR}(i_1, 1) \quad (4) \end{array} \right.$$

The computation is depicted in the block diagram of Figure 3.1. Two quantities, $(i_1 + i_2)$ and $(i_1 \times i_2)$, are common to the construction of all four considered logic functions, which differ only in the coefficients combining these two quantities and in the final sign. Assuming that the inputs i_1 and i_2 take on either 0 or 1 unit of concentration (signifying Boolean 0 or 1 logic value, respectively), one can verify that the four equations correspond to the four intended logic interpretations. In essence, despite the interpretation of gate output is still Boolean, the logic operations are achieved through arithmetic over *reals*, i.e., some form of analog computation, which can be more economical in species requirements than the digital counterpart [20], while also more compatible with the nature of biomolecular reactions used for

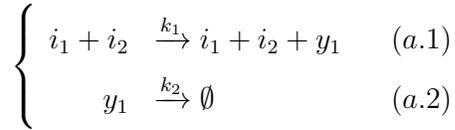
3.2. Reconfigurable logic circuit synthesis



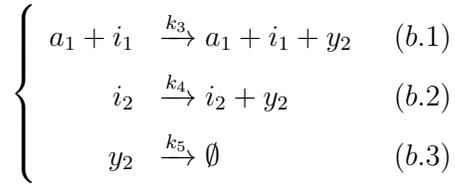
implementation. Notice that the definition of unit concentration is relative, and 0 and 1 units of concentration do not need to be exact; slight deviations in concentration from 0 and 1 are immaterial to the correctness of the interpretation.

Below we show how to implement the above four equations in terms of biochemical reactions. Essentially the four equations are *implemented* by the same set of reactions such that the output value of a configured logic unit coincides with the concentration of some designated species *at equilibrium* in the reactions. According to the block diagram of Figure 3.1, the set of biochemical reactions is comprised of four groups:

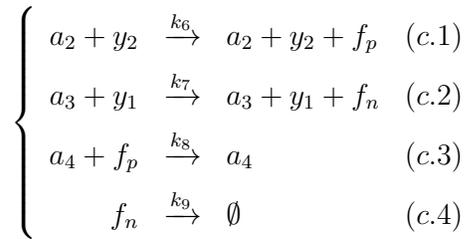
(a) Reactions implementing Submodule 1 in Figure 3.1:



(b) Reactions implementing Submodule 2 in Figure 3.1:



(c) Reactions implementing linear combination:



3.2. Reconfigurable logic circuit synthesis



(d) Reaction implementing output aggregation:



Given the above reactions, we are concerned with their equilibria that we analyze as follows. Reactions (a.1) and (a.2) at equilibrium induce $y_1 = (k_1/k_2)(i_1 \times i_2)$ since

$$\frac{dy_1}{dt} = k_1 i_1 i_2 - k_2 y_1 = 0.$$

Reactions (b.1), (b.2), and (b.3) at equilibrium induce $y_2 = (k_3 a_1/k_5) i_1 + (k_4/k_5) i_2$. Note that, in reaction (b.1), a_1 serves as an auxiliary species, whose purpose is to discharge the stringent rate matching that requires $k_3 = k_4$. With the presence of species a_1 , the constraint on the relation between k_3 and k_4 becomes $k_3 a_1 = k_4$, which can be easily satisfied since a_1 is a species with its concentration tunable externally. That is, we let $a_1 = k_4/k_3$. Reactions (c.1), (c.2), (c.3), and (c.4) at equilibrium induce $(k_8 a_4) f_p - (k_9) f_n = (k_6 a_2) y_2 - (k_7 a_3) y_1$. Similarly, a_2 , a_3 , a_4 are auxiliary species whose concentrations can be controlled externally. Specifically, we let $a_4 = k_9/k_8$, and let the concentrations of a_2 and a_3 be determined depending on the intended logic function (to be discussed). Effectively, species a_2 and a_3 serve as control knobs for function configuration. Finally, assuming K much larger than other rate constants k_1, \dots, k_9 , reaction (d.1) enforces one of output species f_p and f_n to have concentration 0 and the other to have concentration $|f_p - f_n|$.

By the above reactions, the function of a configurable logic unit can be altered by controlling the concentrations of species a_2 and a_3 . Specifically, to configure an AND function, we set $a_2 = 0, a_3 = (k_2 k_9)/(k_1 k_7)$ so that at equilibrium the output f_n equals $\text{AND}(i_1, i_2)$. To configure an OR function, we set $a_2 = (k_5 k_9)/(k_4 k_6), a_3 =$

3.2. Reconfigurable logic circuit synthesis

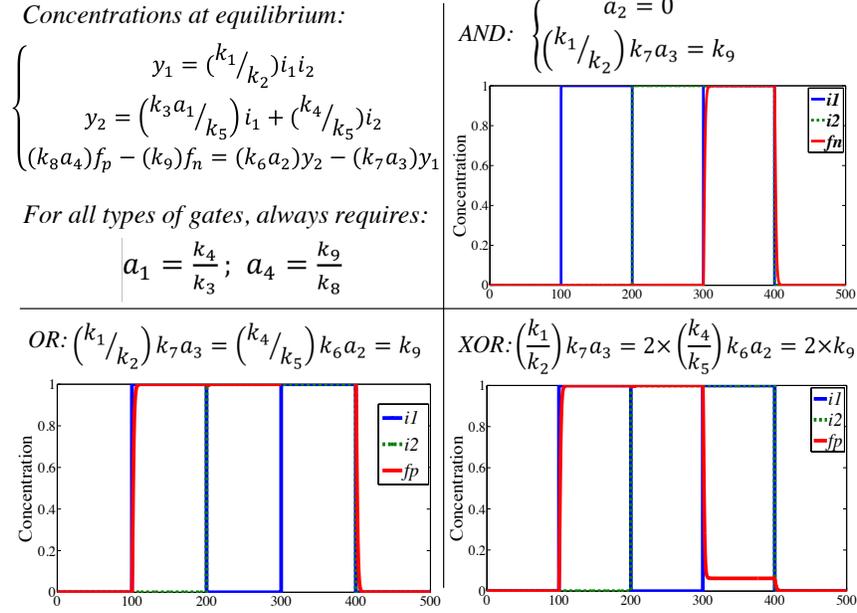


Figure 3.2: Concentration settings and simulation results of the reconfigurable logic unit.

$(k_2k_9)/(k_1k_7)$ so that at equilibrium the output f_p equals $\text{OR}(i_1, i_2)$. To configure an XOR function, we set $a_2 = (k_5k_9)/(k_4k_6)$, $a_3 = 2(k_2k_9)/(k_1k_7)$ so that at equilibrium the output f_p equals $\text{XOR}(i_1, i_2)$. On the other hand, NOT function can be built from XOR. Therefore once inputs i_1, i_2 are assigned to their respective 0 or 1 values, the output converges to 0 or 1 automatically when the above reactions reach their equilibria. Figure 3.2 summarizes the concentration requirements and shows the simulation results under input sequence $(i_1, i_2) = (0, 0), (0, 1), (1, 0), (1, 1), (0, 0)$ in a time separation of 100 units.

To make the module even more *realistic* for implementation, auxiliary inputs a_1, a_4 are introduced to bypass the unrealistic rate constant matching. This is based on the fact that apart from rate constant k , reactants' concentrations also influence the rate of a reaction in a predictable manner. So instead of relying on the nearly impossible *rate constant* matching to achieve the real goal of *reaction rate* matching,

3.2. Reconfigurable logic circuit synthesis

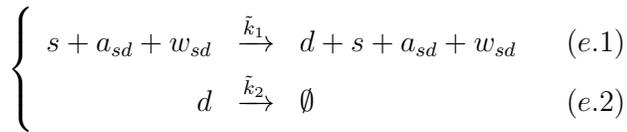


the concentrations of auxiliary species as reactants can be leveraged to achieve the desired matching.

3.2.2 Configurable interconnects

In addition to the four auxiliary input species, each configurable logic unit has two input i_1, i_2 and two output f_p, f_n ports. These ports allow interconnects among multiple configurable logic units, and thus allow arbitrary composition of logic units to realize any Boolean function. We show how an interconnect can be made configurable as follows.

To have a configurable connection between a source port/species s and a destination port/species d , we introduce a unique wiring species w_{sd} for the pair such that s and d are *connected* (i.e., $[d]$ stabilizes to $[s]$ with negligible delay) when w_{sd} is of value 1 (one unit concentration) and *disconnected* (i.e., $[d]$ resets to 0 regardless of $[s]$) if w_{sd} is of value 0 (zero concentration). The reactions that fulfill this connection are:



where a_{sd} is an auxiliary species making $a_{sd} \times \tilde{k}_1 = \tilde{k}_2$ to discharge the need of rate matching of $\tilde{k}_1 = \tilde{k}_2$, and the second reaction serves to reset the destination species d to 0.

Notice that, unlike the well isolation of a signal in electrical circuits, a signal/species in a biochemical circuit without compartmental isolation is globally observable by all reactions. Therefore, instead of using the same information-carrying

3.2. Reconfigurable logic circuit synthesis



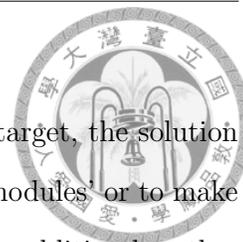
medium (such as voltage or current) for all signals, it is necessary for each signal to be realized by a unique species.

Note also that the *retroactivity* issue, similar to the *loading* effect in electrical circuits, is overcome in our construction by two means. First, the amount of an up-stream species is not affected by composing it with a down-stream species. For example, in reaction (e.1), up-stream species s appears both as a reactant and a product with the same stoichiometric amount. Hence the amount of s remains intact under the presence of reaction (e.1) for the creation of down-stream species d . The same principle is applied to retain the amounts of species $i_1, i_2, a_1, a_2, a_3, a_4$ in the reactions (a.1), (b.1), (c.1), (c.2). Second, we sustain the concentration of a species that can be consumed or produced by some reactions at its intended value based on equilibrium. For example, the concentrations of f_p and f_n remain at their equilibrium values due to the fact that the equilibria of y_1 and y_2 are ensured by reaction groups (a) and (b) since no other reaction involves y_1 and y_2 . Hence in the equilibrium equation $(k_8 a_4) f_p - (k_9) f_n = (k_6 a_2) y_2 - (k_7 a_3) y_1$, the right-hand side is a constant and so are the values of f_p and f_n on the left hand side. (Species a_2, a_3, a_4 have determined constant concentrations.) Thereby our established modularity and composability ensure robust system construction.

3.2.3 Retroactivity resistance

The *retroactivity* issue arises either because the species representing signal at an input port induces *loading* effect by consuming the species representing the connected port(s) upstream, or because the species representative of an output port is consumed by the read-out process of connected port(s) downstream. which can also be regarded as a reversed information flow that disrupts the intended functions by creating

3.2. Reconfigurable logic circuit synthesis



unexpected feedbacks (Figure 3.3). Depending on which side to target, the solution focuses on either to avoid consuming species from ports of other modules' or to make each module's output value immune to output species removal or addition by other module's reactions. Although it is enough to address one side when the system is built of a single type of module, our module incorporates both solutions into the design for future extensibility to accommodate additional modules.

Among the two main lines of strategies to prevent retroactivity from changing the system's behavior, the more "active" one focuses on the module's input side – it attacks the source of the problem by reading the upstream module's output without changing the species' concentrations. This is realized in all our reactions containing input species i_1, i_2 , and auxiliary species a_1, a_2, a_3, a_4 ; the species' concentrations are not changed by the reactions. In a sense, the input species act more like catalysts whose concentrations are not influenced by reactions.

The more "passive" solution focuses on the output side. It addresses the problem: *if a downstream device, apart from reading the output's value, also removes or produces output species of the module at hand, how can the module keep output concentration at its supposed value?* Our method is based on chemical equilibria. Reaction groups (a) and (b) make sure y_1 and y_2 remain at their equilibrium values, and since no other reactions can change their values, the right hand side of equilibrium relation established by group (c) $(k_8 a_4) f_p - (k_9) f_n = (k_6 a_2) y_2 - (k_7 a_3) y_1$ remains constant independent of downstream behavior. Thus the left hand side also remains the correct value even under the influence of retroactivity.

3.2. Reconfigurable logic circuit synthesis

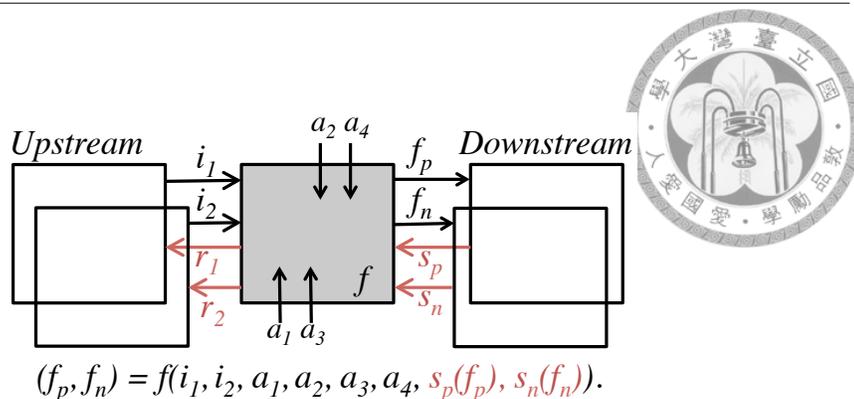


Figure 3.3: Retroactivity can be seen as a “reversed” flow from downstream to upstream component. The flow destroys modularity as it actually can create feedback that makes the module’s output(s) iteratively depend on the output’s value, which can lead to unpredictable and/or unstable behavior. For example, the output of central module depends on retroactivities s_p, s_n from downstream modules, but s_p (resp. s_n) is also a function of the module’s output f_p (resp. f_n). The interdependency actually establishes a feedback relation.

3.2.4 Multiplexer-based structure

With our main focus remained on reconfigurable logic gate, here we briefly touch on the biochemical-reaction implementation of another common *programmable* logic devices, the multiplexer, and discuss the potential benefits of this alternative approach.

Multiplexer can be regarded as a guard that allows exactly one selected data to pass through. According to the value of the selection signal, only one input is relevant at any time point. Figure 3.4(a) shows the simplest 2-to-1 example, in which the *level* of the *control* signal

From this perspective, multiplexer can serve as a natural interface between *digital* control signals (which may be the activated/inactivated signals like those in gene regulatory network, or communicating signals initiated by electronics, etc.) and *analog* information flow (such as the concentration of biochemical species, the wide varieties of real-value-weighted signals, or electrical current, etc.).

In fact, when it comes to biochemical implementation, adopting the multiplexer

3.2. Reconfigurable logic circuit synthesis

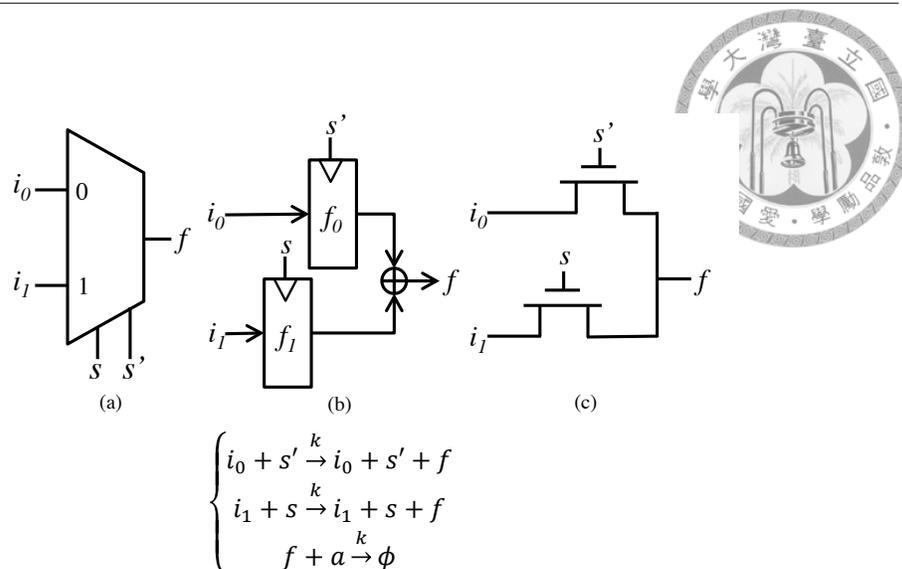


Figure 3.4: (a) A 2-to-1 multiplexer, (b) a parallelized view of its function, and its (c) direct mapping to pass-transistor logic. The reactions listed at the bottom implement the multiplexer function.

structure can reap much benefit due to reasons closely related to the non-locality of chemical species and reactions—especially when the number of control signals are few comparing to the controlled information flow. First, the reactions for multiplexer can be optimized using the distributed nature of chemical species, so the implementation becomes simpler than directly using reactions to build logic gates that can then be assembled into a multiplexer. Second, only two species are required for each controlling variable no matter how many multiplexers it controls. Another interesting point to note is that: chemical reaction implementation takes full advantage of the fact that each node in BDD represents a boolean function – answers of multiple functions can be obtained simultaneously by accessing the concentrations of corresponding species.

Considering that at any time point, only one input is relevant, combined with the fact that the functions realized with different selected signals are symmetric, we adopt dual-rail representation for selection signals while preserving single-rail implementation for all other signals. However, the *dual-rail* here is more general

3.2. Reconfigurable logic circuit synthesis



in that mutual exclusion is no longer a vital requirement because the two reactions respectively responsible for either selection cases ($s = 1$ or $s' = 1$) can coexist safely side-by-side (though the multiplexer would then become an adder). Apart from reducing the required number of wires, another advantage of this implementation is its ability to propagate not only boolean variables, but also data of *positive real* value.

The electronic analogy of this implementation is the *Pass Transistor Logic (PTL)*. In PTL, primary inputs drive gate terminals as well as source-drain terminals, in contrast to static CMOS, whose primary inputs drive only gate terminals. The close resemblance allows us to benefit from the synthesis and optimization methods already developed for PTL [10, 77]. Besides, our *equilibrium*-based design is capable of resisting signal deterioration that comes with long signal transmission chain—thus literally removes the length constraint posed on pass-transistor chain design, which cannot be too long to guarantee signal's integrity.

Reactions used to implement a multiplexer with function:

$$f = \begin{cases} i_0, & \text{if } s = 0 \text{ and } s' = 1 \\ i_1, & \text{if } s = 1 \text{ and } s' = 0 \end{cases}, \quad i_0, i_1 \in R; s, s' \in \{0, 1\}.$$

are listed in Figure 3.4.

A point worth noting is that: if the mutual exclusive constraint on s and s' is not met, the design actually becomes a *tri-state buffer* when $s = s' = 0$, and a *real-valued* adder that implements $i_1 + i_2 = f$ when $s = s' = 1$.

3.2. Reconfigurable logic circuit synthesis



From Boolean logic specification to CRNs

Binary decision diagrams (BDD) can be directly translated into digital circuits by substituting each node by a multiplexer controlled by the variable of the node. So circuits composed of multiplexers can be seen as direct mappings from BDDs, so it is safe to say that an optimized multiplexer circuit comes from an optimized BDD. Assume that all variables are not redundant, the number of multiplexers and wirings both increase with the number of nodes in the BDD used for mapping. The distributed nature of chemical species and reactions remove structural concerns related to manufacturability in traditional circuit design, so the optimization objective can be safely concluded as one of node reduction. *Free BDD* and *Decomposed BDD* are two existing optimized BDD forms with mature generation heuristic suitable for our purpose here.

Another optimization possibility is to incorporate multiple desired functions into multiple nodes in a shared BDD. Because our reaction design essentially rules out the retroactivity effect, the answers can be read out directly from the concentration of each corresponding node's f species, as shown in Figure 3.4.

3.2.5 Case study

A microRNA (miRNA) is a small, highly conserved non-coding RNA that involves in almost every cellular process and down-regulates gene expressions through partial base-pairing with its (multiple) messenger RNA (mRNA) targets. Inappropriate miRNA expressions have been linked to the regulation and progression of a wide range of diseases [12], such as numerous cancers, cardiovascular, neurological, immunological, and metabolic diseases. Early onset of those diseases can be detected by monitoring

3.2. Reconfigurable logic circuit synthesis



changes in miRNA expression levels. Due to the *partial* base-pairing during target recognition, the regulation relation between miRNAs and mRNAs is *many-to-many*. As a result, diagnosis of certain diseases may involve multiple miRNAs and complex decision conditions, which may be expressible in Boolean formulae.

For potential implementation of our proposed biochemical reactions, there is recent demonstration of oligonucleotide AND-gates that can respond to specific miRNA inputs in live mammalian cells [37]. Moreover, *DNA strand displacement* [72, 79] has been successful in implementing various chemical reaction networks. These techniques may bring promise to the feasibility of conducting Boolean operations on miRNA inputs, recognizing endogenous miRNA expression patterns, and generating different oligonucleotide outputs correspondingly to manipulate miRNA levels for therapeutic purposes.

As reconfigurable circuitry may conduct different computation tasks utilizing the same set of reactions, it may realize different diagnostic and therapeutic strategies whichever one is needed. As a thought example, we consider function switching between two diagnostic-therapeutic specifications expressed in two Boolean expressions f_1 and f_2 :

$$f_1 = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4)$$
$$f_2 = (x_1 \wedge x_2 \wedge x_3) \vee (\neg x_2) \vee (\neg x_1 \wedge x_4)$$

where \wedge , \vee , \neg are Boolean connectives conjunction (and), disjunction (or), and negation (not), respectively. Imagine that each variable x_i represents a distinct type of miRNA related to the diagnostic tasks at hand. Let f_1 and f_2 encode the therapeutic actions corresponding to the diagnostic tests of diseases A and B, respectively. When disease A (respectively B) is in consideration, the reconfigurable

3.2. Reconfigurable logic circuit synthesis

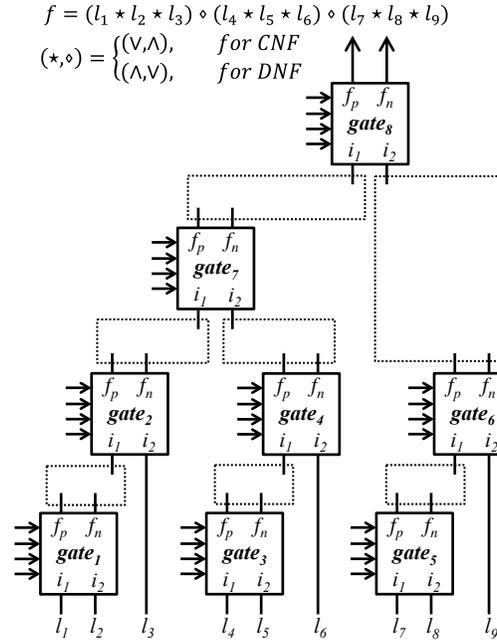


Figure 3.5: A circuit diagram implementing functions f_1 and f_2 . Gates 1 to 6 form the first level of logic; gates 7 and 8 form the second logic level. The four arrows pointing *into* each gate are the *auxiliary inputs*; the dashed rectangles indicate places for wirings: wiring from f_p is established if the source gate implements function *OR*, and f_n in the case of *AND*.

circuitry implements f_1 (respectively f_2) function. The function output may be coupled with some miRNA whose expression level is to be raised for disease treatment.

The static group of modules, which defines the unchanging set of reactions used, is shown in Figure 3.5. Note that the *diagnosis conditions* compatible with the structure are *not limited to* f_1 and f_2 , but *all* CNF and DNF functions that can fit into the listed general form, i.e. having at most 3 literals in a clause/cube, and a maximum of 3 clauses/cubes. In this circuit setting, literals are mapped according to the general form described on the top of Figure 3.5. When a clause/cube contains less literals than the upper limit, the literals left are set to the non-controlling value of that gate. For example, in the case of f_1 : $(l_1, l_2, \dots, l_9) = (x_1, x_2, 0, \neg x_1, x_2, \neg x_3, x_3, x_4, 0)$.

3.2. Reconfigurable logic circuit synthesis

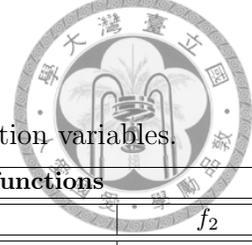


Table 3.1: Correspondence between circuit signals and function variables.

circuit		functions		circuit		functions	
wires	f_1	f_2	wires	f_1	f_2		
f_{p1}	$x_1 + x_2$	$(\neg x_1)(\neg x_2)$	f_{n1}	$\neg x_1 + \neg x_2$	$x_1 x_2$		
f_{p2}	$x_1 + x_2$	$(\neg x_1)(\neg x_2)$	f_{n2}	$\neg x_1 + \neg x_2 + \neg x_3$	$x_1 x_2 x_3$		
f_{p3}	$\neg x_1 + x_2$	$(x_1)(\neg x_2)$	f_{n3}	x_2	$\neg x_2$		
f_{p4}	$\neg x_1 + x_2 + \neg x_3$	$(x_1)(\neg x_2)(x_3)$	f_{n4}	x_2	$\neg x_2$		
f_{p5}	$x_3 + x_4$	$(\neg x_3)(\neg x_4)$	f_{n5}	$x_1 + \neg x_4$	$\neg x_1 x_4$		
f_{p6}	$x_3 + x_4$	$(\neg x_3)(\neg x_4)$	f_{n6}	$x_1 + \neg x_4$	$\neg x_1 x_4$		
f_{p7}	$\neg x_1 + \neg x_2 + \neg x_3$	$x_1 x_2 x_3$	f_{n7}	$(x_1 + x_2)(\neg x_1 + x_2 + \neg x_3)$	$(x_1)(\neg x_2)(x_3)$		
f_{p8}	$\neg f_2$	f_2	f_{n8}	f_1	$\neg f_1$		

A schematic diagram implementing the above two functions is shown in Figure 3.5, where the gates correspond to the configurable logic units introduced in Section 3.2.1, the four side-inputs to a gate indicate the auxiliary inputs, and the dashed boxes correspond to the configurable interconnects. For simplicity, here the configurability of interconnects is only limited to certain port to port connections. To implement functions f_1 and f_2 on the circuit shown, the inputs $(l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8, l_9)$ are assigned $(x_1, x_2, 0, \neg x_1, x_2, \neg x_3, x_3, x_4, 0)$ for f_1 , and $(x_1, x_2, x_3, \neg x_2, 1, 1, \neg x_1, x_4, 1)$ for f_2 . Gates 1 to 6 implement the part of logic *inside* the parentheses in the formulae of f_1 and f_2 , while gates 7 and 8 implement the logic operations that connect between parentheses.

Table 3.1 gives the Boolean functions with evaluated results presented at their corresponding ports in the circuit shown in Figure 3.5, where f_{pi} and f_{nj} indicates respectively the f_p port of gate i and the f_n port of gate j .

The above reconfigurable circuit is simulated using BIOCHAM [22]. The input waveforms and resultant output waveforms are shown in Figure 3.6 and Figure 3.7, respectively. The configuration switches from function f_1 to function f_2 at time $t = 110$. After connection configuration is established at $t = 10$, the input values change every 50 time units, at $t = 60, 110, 160$, with input sequence

3.3. Level-based neuromorphic computation

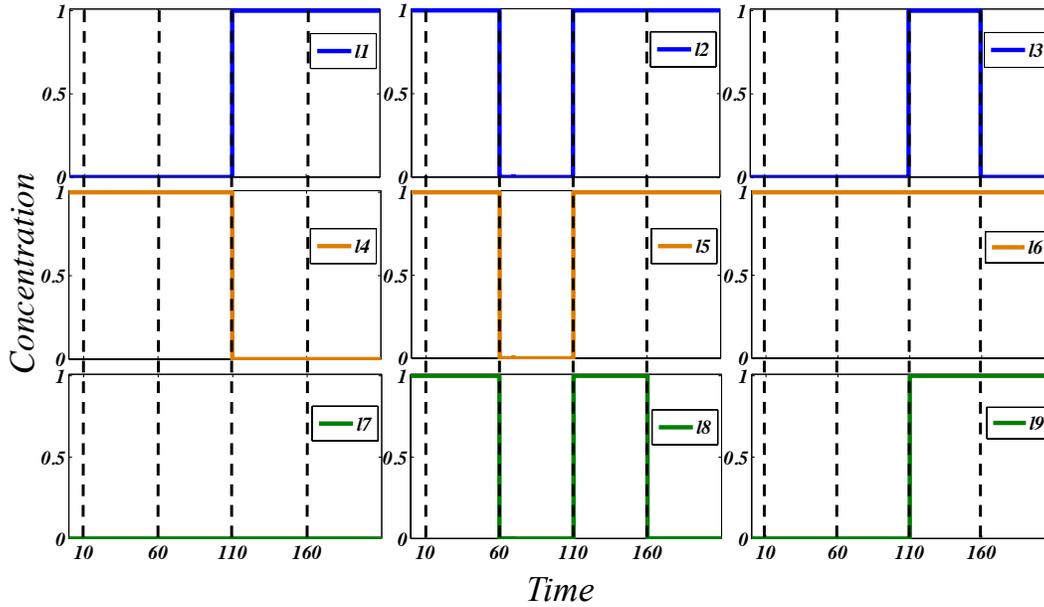
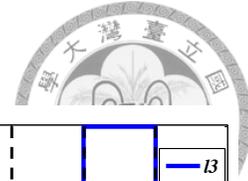


Figure 3.6: Waveforms of inputs l_1, l_2, \dots, l_9 . The four vertical dashed lines indicate important time points: (1) $t = 10$: connect all wires; (2) $t = 60$: change primary input values; (3) $t = 110$: change implemented function from f_1 to f_2 , and change the value of PIs; (4) $t = 160$: change the value of PIs again.

$(x_1, x_2, x_3, x_4) = (0, 1, 0, 1), (0, 0, 0, 0), (1, 1, 1, 1), (1, 1, 0, 0)$, which imitates the change of miRNA expression patterns. The waveforms of l_1, \dots, l_9 in response to the input sequence is shown in Figure 3.6. The waveforms of f_1 and f_2 are shown in Figure 3.7, which imitate the therapeutic responses to diseases A and B, respectively.

3.3 Level-based neuromorphic computation

Nature's competence in operating complex systems and solving a wide range of real world problems may be a strong hint that instead of pursuing the long-dominant algorithmic approach on Von-Neumann architecture, the more effective, or even the only feasible way to embed those capabilities into engineered systems is to adopt

3.3. Level-based neuromorphic computation

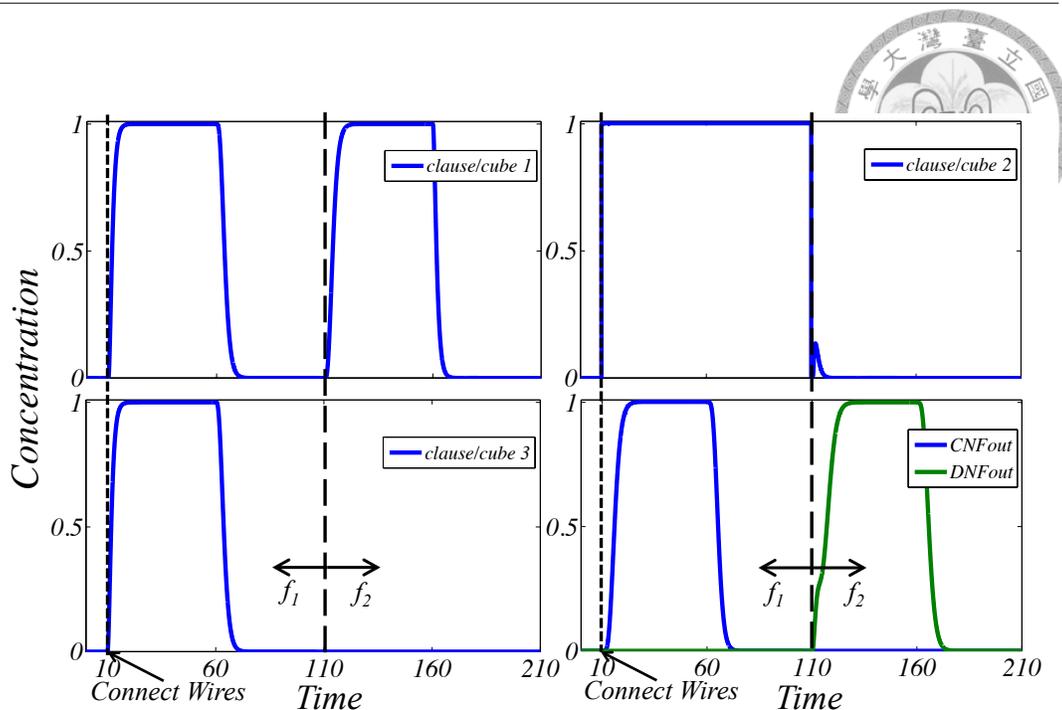
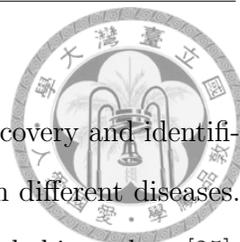


Figure 3.7: Waveforms of outputs f_1 and f_2 . Simulation result based on the input values given in Figure 3.6.

the structure and dynamics directly from natural designs. During the accelerating advance in systems and synthetic biology, it was identified before long that the two crucial features—*adaptation* and *evolution*—which allow living organisms to achieve certain goals *reliably* despite the always-changing environment, are essentially the biological solution, polished by nature from real-world applications, to the coveted engineering pursuits of *spontaneous reconfiguration and optimization* of a system’s functionality. Inspired by the observation that many tasks that demand substantial engineering efforts—such as perception, association, and non-linear control—place no obvious obstacles to even simple living organisms, *neural system* has become the imitation target while building systems whose value depends largely on their ability to adapt efficiently to the environment, possibly *without human intervention*. This is exactly the case for many synthetic biological pursuits toward realizing bio-computing systems.

3.3. Level-based neuromorphic computation

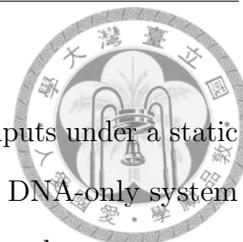


One important example in medical applications follows the discovery and identification of physiological biomarker [57, 76] patterns associated with different diseases. Generally, the patterns involve non-trivial relations between multiple biomarkers [25], and are often obscured by environmental *noises* and *variations* under different scenarios. Given all the uncertainties, *self-adapting bio-computation* is an indispensable component for the engineered system to decide whether certain disease-implied pattern is *really* present [20, 37], before appropriate signaling and on-site immediate intervention based on analyzed results are possible.

A closer look at the topic makes it clear that at least in two ways will bio-computation systems benefit from including neural-network-like self-adaptation: First, because both the engineered system itself and its external environment are of biochemical nature, unpredictable variations in the system's behavior and the environment's conditions are the norm. Self-adaptation allows the system to *maintain correct functionality* under varying conditions and to compensate for the system's own deviation from original design. Second, different scenarios may require different functions for targeted outcome (ex. different therapies are required for different diseases detected on-site); if the system is capable of learning the selection criteria and making decisions accordingly, multiple functions devised for different scenarios can be incorporated into a single bio-computation system to reduce redundancy, while the appropriate one will be autonomously selected by the system based on the scenario occurred.

However, molecular-based implementation of neuromorphic computational system, especially one with built-in learning ability, is still lacking. Only until recently, [63] presented the first biomolecular implementation of a neuron and based upon it neural networks using DNA molecules. However, its use-once architectures proposed

3.3. Level-based neuromorphic computation



prohibit the system from correctly processing series of changing inputs under a static function. Besides, while the theoretical possibility of building a DNA-only system with dynamic behavior was briefly mentioned, no real design of such a system was presented. In fact, considering that before neuron-based brains had evolved, there must have existed some biomolecular-based mechanism responsible for the intelligent behavior that ensures survival of organism without a neural system, the pursuit of realizing self-adapting ability in designed biomolecular reactions may even seem natural. For example, the single-celled paramecium makes use of chemical processes that affect the electrical potential across their membrane and modify the shape of the constituent proteins. Chemical processes also regulate the behavior of multicellular sponges; their contractile cells can absorb nutrients from water pumped through the body according to chemical signals from environmental stimuli.

In this subsection, we would begin our pursuit of embedding the neuromorphic computation into the more ubiquitous biochemical reaction from the more abstract *level*-based neuron model. We present in this and the next subsection two chemical reaction construction of reconfigurable artificial neural networks. In both cases, module-based architectures are adopted, with each module corresponding to a neuron of respective model of choice; each wiring to a direct wire-like, or a synapse-like interconnection between two neurons.

3.3.1 Artificial neural network and adopted neuron model

The key element of neuromorphic computation paradigm lies in the structure of the information processing system. Units implementing identical analog computations update their states *continuously* and *asynchronously* based on simultaneous interactions spanning through multiple units. It can thus be regarded as a large number of

3.3. Level-based neuromorphic computation

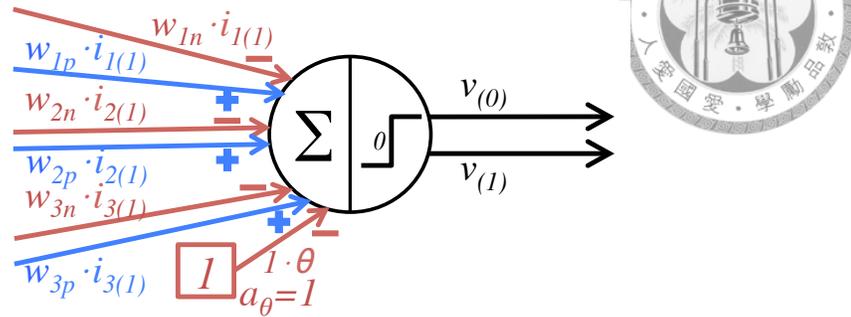
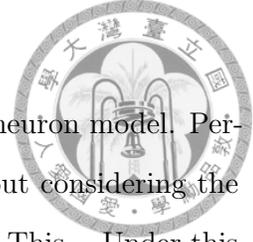


Figure 3.8: A perceptron with three inputs.

highly interconnected units (each with limited processing ability) working in unison to solve specific problems. From the inference of comparative biology, the morphology and behavior of individual neurons are very similar across both animal species and evolution history. Considering the substantially different level of cognitive ability demonstrated in today's animals and that of primitive animals, it is reasonable to say that evolution of the brain lies mostly in the architectural level, i.e. the pattern of connectivity established between neurons. Instead of establishing an exhaustive modeling of every working details, *artificial neural networks* [65] attempt to extract part of the strength from biological neural networks by modeling the most crucial mechanisms and abstracting out other details. The feedforward network [35] and the Hopfield network [42] are among the most popular and well-explored architectures; more detailed explanation will come in later Section 3.3.6 with our biochemical reaction based implementations.

As for the artificial *neuron* serving as processing unit in the network, there exists various models that serve different purposes and by necessity represent a caricature of a biological neuron in some context. Different models are characterized by their different levels of abstractions and choices on which properties of biological neurons to include.

3.3. Level-based neuromorphic computation



In this section, we adopt the *perceptron* [66] as our default neuron model. Perceptrons based their computation on the *level* of inputs, without considering the input's timing profile as is the case in spike-based computation. This... Under this model, the output of a neuron with n inputs and threshold: $i_1, \dots, i_n, \theta \in \mathbb{R}^+ \cup \{0\}$ is determined by the activation function

$$f(\vec{i}) = \begin{cases} 1, & \text{if } \sum_{j=1}^n w_j i_j \geq \theta, \\ 0, & \text{otherwise,} \end{cases} \quad (3.1)$$

where $w_j \in \mathbb{R}$ is the corresponding synaptic weight of input i_j . That is, a perceptron decides its output by comparing the weighted sum of its inputs with its threshold value, producing a 1 if the weighted sum exceeds the threshold, and 0 otherwise.

3.3.2 General architecture of neuromorphic FPGA

Similar to FPGAs, our proposed neuromorphic architecture consists of reconfigurable neuron modules and reconfigurable interconnects.

In this section, we present a chemical reaction-based neuromorphic architecture that allows dynamic reconfiguration by controlling the concentrations of some preassigned species. Each *module* implements a neuron, and each interconnect between modules being the *abstraction* of a synapse, which actually becomes more like a direct wiring in circuits topped with weighting. The reconfigurability of the system lies in controllable weights of all interconnects and independently tunable thresholds of each neuron. The simple yet powerful *perceptron* neuron model introduced in Section 3.3.1 is adopted. In this section, *module* and *perceptron* will be used interchangeably when describing our proposed FPGA-like architecture.

3.3. Level-based neuromorphic computation

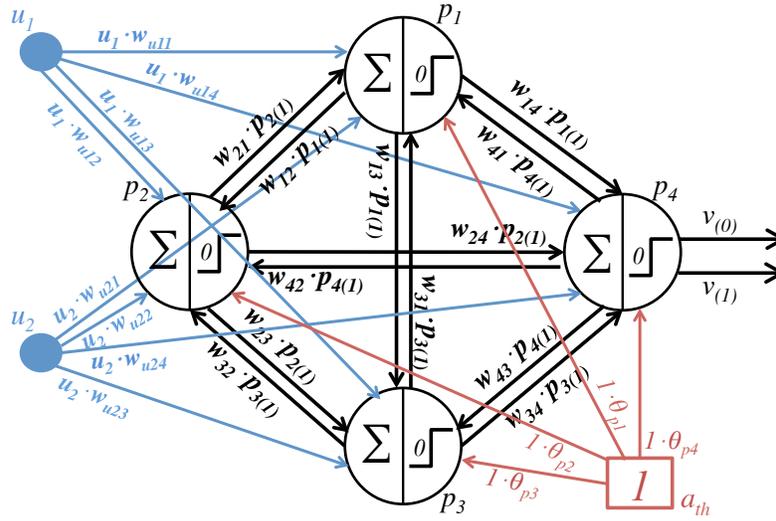


Figure 3.9: A complete digraph network with four perceptrons.

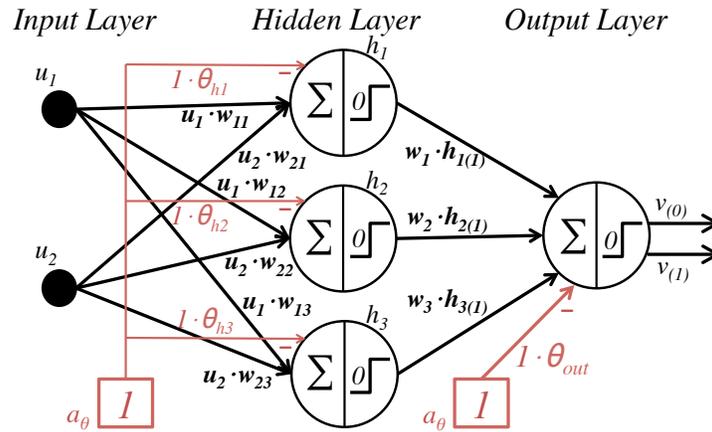


Figure 3.10: A feed-forward network with one hidden layer.

As will be shown in Section 3.3.4, our proposed architecture has the capacity allowing interconnects between all pairs of modules in both directions (i.e., a *complete digraph*) to be established simultaneously, and can thus accommodate any arbitrary topology. Considering that the *structure* of a neural network, especially the interconnecting pattern, is also crucial for selecting the suitable tasks to be performed, and the type of learning algorithms that can be effectively implemented by the network, the flexibility provided by the capacity of our proposed architecture is valuable

3.3. Level-based neuromorphic computation



However, it is worth noting that when resource requirement is the more important concern, limiting the design space to some certain topology allows for early optimization toward fewer reactions and species while maintaining the correctness of the function. For example, when the structure of the network to be implemented is not known in advance, all pairs of interconnects are possible, thus every single pair calls for its unique set of reactions and species that *may* be used later. Once a species is assigned to an interconnect, it becomes observable to the whole system and cannot be used again in any other interconnects to prevent the mixing up of different signals. We say that the assigned species (and its related reactions) are *reserved* for the *exclusive* use of that possible interconnect.

On the contrary, if the topology is known beforehand to be *feedforward*—in which case no interconnect is allowed between modules from the same layer, while each module is connected to all modules from the two neighboring layers—we do not have to *reserve* reactions and species that otherwise would have been assigned in vain for these impossible interconnects. The early knowledge of network topology thus can lead to reduced resource requirement. In fact, the reduction can be substantial even for relatively small systems. To give a feel of the extend of reduction, here we give a quantitative comparison between a *complete digraph* network and a *feedforward* network, as shown respectively in Figure 3.9 and Figure 3.10:

Both networks are composed of 4 neurons in total, with 2 inputs and 1 output. First we consider the requirements shared by both systems:

- An auxiliary species a_θ shared by all neurons with constant unit concentration;
- One unique species for each input value;
- As will be shown in Section 3.3.3, each neuron, before taking inputs into

3.3. Level-based neuromorphic computation



account, requires 6 species and 7 reactions ((I.3), (II.1)~(II.6)).

(a)~(c) add up to 27 species and 28 reactions which forms the minimum requirement for all 4-neuron networks of the same input/output count.

The major difference between the two networks lies in the number of *interconnects*; each interconnect asks for another 2 species and 2 reactions. Note that although in both Figure 3.9 and Figure 3.10, only one edge is shown to represent an interconnect for clarity, each is actually realized as dual-rail signals to accommodate both positive and negative weights because concentrations can never be negative. Generally a complete digraph with n_i inputs and p perceptrons has $n_i \times p + 2 \times C_2^p$ interconnects, which leads to 20 in this example; compare with the 9 interconnects involved in feedforward network. To sum up, (*number of reactions*, *number of species*) equals (68, 67) for complete digraph; (45, 46) for feedforward network.

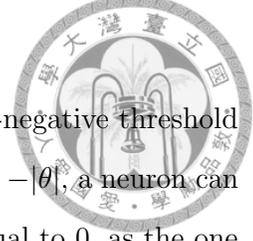
The optimization opportunity can be of great value for biochemical system design because of the stringent constraints on species to ensure practical implementation.

3.3.3 Neuron module

As can be observed from Eq. 3.1, the abrupt change in output given a continuous input space makes it more appropriate to treat perceptron as a bistable switch with transition happening when the weighted sum equals threshold value, rather than to explicitly construct a formula that computes output value from input values.

To represent a real-valued signal x , two species x_p and x_n are designated with $x = [x_p] - [x_n]$, similar to [60]. When the input weight w_i of a neuron is positive (resp. negative), $[w_{ip}]$ (resp. $[w_{in}]$) is set to the absolute value of positive (resp. negative)

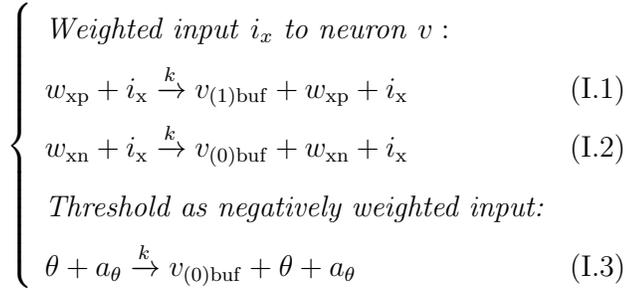
3.3. Level-based neuromorphic computation



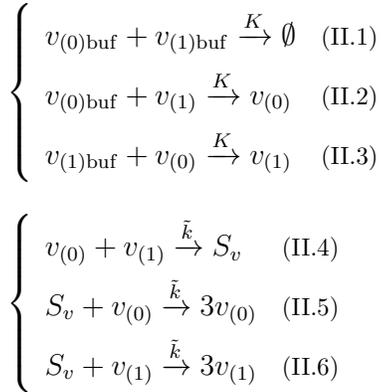
weight and $[w_{\text{in}}]$ (resp. $[w_{\text{ip}}]$) is set to zero. By interpreting non-negative threshold value θ as an auxiliary input $a_\theta \equiv 1$ with negative weight equal to $-|\theta|$, a neuron can always be transformed into an equivalent one with threshold equal to 0, as the one illustrated in Figure 3.8. Therefore it suffices to implement a single bistable reaction system for a neuron whose output toggles at the *zero* threshold point.

We use the three-input neuron as depicted in Figure 3.8 to explain the two main components of CRN implementation listed below:

(I) To compute the weighted sum (represented by the generation rate difference between molecules $v_{(1)\text{buf}}$ and $v_{(0)\text{buf}}$) of inputs (including i_x and the threshold input a_θ) for neuron v , we rely on the following reactions, with $x = 1, 2, 3$ for three inputs:

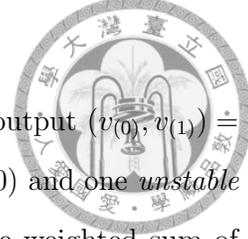


(II) To determine whether the weighted sum exceeds 0 (i.e., whether $[v_{(1)\text{buf}}] - [v_{(0)\text{buf}}] \geq 0$), we depend on the bistability created with the following reactions.



We start our discussion from (II). Reactions (II.4)~(II.6) create a bistable sys-

3.3. Level-based neuromorphic computation



tem [47] with two *stable* steady states (represented by dual-rail output $(v_{(0)}, v_{(1)}) = (0, 1)$ signifying neuron output 1; $(1, 0)$ signifying neuron output 0) and one *unstable* steady state (at $(v_{(0)}, v_{(1)}) = (0.5, 0.5)$). To decide whether the weighted sum of inputs is larger than zero (i.e., whether the sum of the positively weighted inputs is larger than the absolute value of the sum of the negatively weighted inputs) and to require $(v_{(0)}, v_{(1)}) = (0, 1)$ (resp. $(1, 0)$) when the sum of the positively weighted inputs is larger (resp. smaller) than the sum of the negatively weighted inputs, we establish the correspondence between $v_{(0)}$ (resp. $v_{(1)}$) and *negatively* (resp. *positively*) weighted inputs by the reactions in (I) and (II.1)~(II.3). It should be clarified that reactions (I.1) and (I.2) are *not* an intrinsic part of the module, but rather their presence depends on the existence of their corresponding *interconnects* between modules. The detailed reactions will be given in Section 3.3.4.

When the weight of the x^{th} input i_x is positive (effectively $w_{\text{xn}} = 0$), only the reaction with w_{xp} involved is activated and thus $v_{(1)\text{buf}}$ is generated at rate $(k \cdot w_{\text{xp}} \cdot i_x)$. For the y^{th} input i_y with a negative weight, the same reasoning applies and $v_{(0)\text{buf}}$ is generated at rate $(k \cdot w_{\text{yn}} \cdot i_y)$. Reaction (I.3) effectively subtracts the threshold value θ from the weighted sum of inputs. With the reactions in (I), the *generation rates* of molecules $v_{(1)\text{buf}}$ and $v_{(0)\text{buf}}$ correspond respectively to the intended sums of the *positively* and *negatively* weighted inputs. Reactions (II.1)~(II.3) then convert the comparison between the *generation rates* of $v_{(1)\text{buf}}$ and $v_{(0)\text{buf}}$ to the comparison between the *concentrations* of $v_{(0)}$ and $v_{(1)}$. Finally, reactions (II.4)~(II.6) enforce the concentrations of $v_{(0)}$ and $v_{(1)}$ at equilibrium stabilize to one of two the stable steady states discussed in the previous paragraph.

Note that the conversion achieved by reactions (II.1)~(II.3) is crucial in *preserving the total number of output molecules* ($[v_{(0)}] + [v_{(1)}]$), so the system does not require

3.3. Level-based neuromorphic computation



constant replenishment of species from outside. The effort not only makes the system more practical, but also avoids deviation of system behavior resulted from inaccurate replenishment. This *conservation* of total number of output molecules is critical for implementing the frequently needed classifiers with precision, especially so when the input space is *continuous*—usually the case in biochemical systems—rather than discrete. Small deviations in the output value of one neuron might be amplified by a larger weighting, spread through the highly connected network, and lead to wrong flips of output values in possibly *multiple* downstream nodes.

To guarantee that the ratio of the positively to negatively weighted sums of inputs is the same as the ratio of the generation rate of $v_{(1)\text{buf}}$ to the generation rate of $v_{(0)\text{buf}}$, all the reactions in (I) would require the same rate constant k . This requirement is unrealistic and can be overcome by our engineered reconfigurability [18]. Because the rate of each reaction in (I) can not only be regarded as a function of k but also as a function of $k \times w_p$, $k \times w_n$, or $k \times \theta$ for species w_p , w_n , or θ *unique* to that reaction, we can relax the original rate constant constraint $k_{(I.1)} = k_{(I.2)} = k_{(I.3)} = k$ to $(k_{(I.1)} \times w'_{\text{xp}}) = (k_{(I.2)} \times w'_{\text{xn}}) = (k_{(I.3)} \times \theta')$, where the primed version w' of w signifies that the value of w' corresponds not exactly to an original input weight as w , but to an input weight adjusted for the purpose of rate matching.

In the proposed design, all the weights and thresholds are mapped to *distinct* biochemical species, allowing each to be controlled directly through the concentration of its corresponding species. In later case studies, we show that neural networks of different structures can be easily constructed by simply “mixing” together the reactions implementing each module and the weighted connections. The biochemical reactions proceed *concurrently* as in real-world neural system.



3.3.4 Programmable interconnect

Once the set of available modules \mathbf{m} are constructed, a directed interconnect from an arbitrary module $m_i \in \mathbf{m}$ to an arbitrary module $m_j \in \mathbf{m}$ with reconfigurable weighting $w_{ij} \in \mathbb{R}$, expressed as: $m_i \xrightarrow{w_{ij}} m_j$ as shown in Figure 3.11, can be *reserved* by adding the following reactions to the existing set of reactions.

$$\left\{ \begin{array}{l} w_{ijp} + v_{i(1)} \xrightarrow{k} v_{j(1)buff} + w_{ijp} + v_{i(1)} \\ w_{ijn} + v_{i(1)} \xrightarrow{k} v_{j(0)buff} + w_{ijn} + v_{i(1)} \end{array} \right.$$

One *directed* interconnect requires the reservation of two reactions: one for the *positively* weighted input and the other for the *negatively* weighted input, due to the natural limitation that reactions and concentrations cannot operate, or take on negative values. Corresponding to the two reactions are two species ($w_{ij(p)}, w_{ij(n)}$) whose concentrations are used to control the weight. Therefore implementing an interconnect costs 2 reactions and 2 species.

As a neural network's strength depends much on the high interconnectivity among neuron modules, apart from the *error propagation* problem that is the target of our proposed solution of molecule conservation (reactions (II.1)~(II.3)), the *loading effect*, or the retroactivity introduced with each additional connection, becomes another important design consideration concerning the system's performance. Our design tackles this problem by making sure the reactions for an interconnect do not alter the equilibrium of the source module: the value of $v_{i(1)}$ reached without downstream interconnects remains unchanged with the existence of interconnect reactions.

Note that the modules on the two sides of the interconnects are not required to be distinct, i.e., it is acceptable to have $m_i = m_j$, and the interconnect is said to be *recurrent*.

3.3. Level-based neuromorphic computation

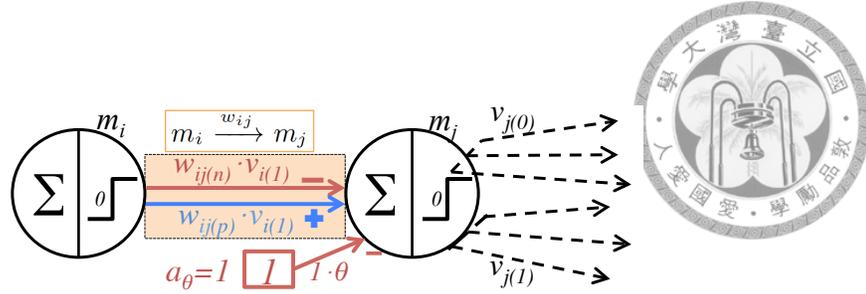


Figure 3.11: interconnects of positive and negative weights.

3.3.5 Resource requirement

A feedforward neural network with x inputs, z outputs (when working as a classifier, it may classify up to 2^z classes in x -dimensional input space [81]), and n hidden layer(s) with y_i nodes in the i^{th} layer requires:

$$[(x + 1) + 6 \times (z + \sum_{i=1}^n y_i)] + [2 \times (xy_1 + \sum_{i=1}^{n-1} y_i y_{i+1} + y_n z)] \text{ species, and}$$

$$[7 \times (z + \sum_{i=1}^n y_i)] + [2 \times (xy_1 + \sum_{i=1}^{n-1} y_i y_{i+1} + y_n z)] \text{ reactions.}$$

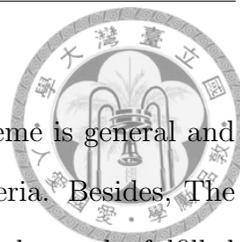
Given any neural network, the mapping of its neurons and interconnects to our architecture is doable in linear time by assigning reaction species in (I.1) (I.2) and (II.1) (II.6) for each interconnect and neuron, respectively.

3.3.6 Case study: classifier synthesis with known criteria

As mentioned in the beginning of Section 3.3, *adaptation* and *evolution* are two crucial features of biological systems, which can lead to solutions to many design challenges met in synthetic biology.

We start from demonstrating the reconfigurability of our neuromorphic architecture with an example of classifier mapping in this case study. As will also be shown

3.3. Level-based neuromorphic computation



along with the example, the proposed systematic mapping scheme is general and can be applied to classifier synthesis for some given set of criteria. Besides, The reconfigurations do *not* require structural changes, i.e., multiple tasks can be fulfilled with the same set of reactions, thus dynamic adaptation is achievable without the demanding process of redesigning and finding new reactions and species.

Without loss of generality while taking biological reality into account, we assume that the elements of inputs are all *non-negative*. The mapping problem is formulated as follows:

Problem Given $(m - 1)$ criteria of the form:

$$\begin{aligned} \langle \textit{criterion} \rangle &::= \langle \textit{inequality} \rangle \\ &| \neg \langle \textit{criterion} \rangle \\ &| \langle \textit{criterion} \rangle \vee \langle \textit{criterion} \rangle \\ &| \langle \textit{criterion} \rangle \wedge \langle \textit{criterion} \rangle \\ \langle \textit{inequality} \rangle &::= \mathbf{w}^T \mathbf{i} \geq k, \text{ where } \mathbf{w} \in R^m, \mathbf{i} \in R_0^+, k \in R. \end{aligned}$$

Each criterion defines the condition for an n dimensional input vector \mathbf{i} to be classified into a corresponding class among a total of m classes in the n dimensional input space.

Map the specified classifier to a set of chemical reaction implementation by determining: (1) the number of modules in each layer; (2) the connection weights; (3) the threshold of each neuron module.

Note that because the chemical reaction implementation of the module and interconnect are both already well-defined in our FPGA-like architecture, and the fact

3.3. Level-based neuromorphic computation



that any *arbitrary* classification, linear or not, can be achieved by some feedforward neural network with one hidden layer as shown in [43], Problem 3.3.6 is actually one of deciding species concentrations in networks with *one* hidden layer.

For the demonstrating example, consider a feedforward network with one input, one hidden, and one output layer as shown in Figure 3.10, which implements a classifier that separates the input space spanned by $u_1, u_2 \in \mathbb{R}^+ \cup \{0\}$ into two classes based on whether the criterion below is satisfied:

$$(5u_1 - u_2 \geq 3) \vee [(-u_1 + 2u_2 \geq 1.5) \wedge (u_1 + u_2 \geq 1.5)]$$

The number of inputs correspond to the dimension of input space. Here, the input layer consists of two inputs u_1 and u_2 . The output layer requires $\lceil \log_2(\text{number of classes}) \rceil$ neurons, which equals 1 in this example. Each neuron in the first hidden layer can define a separating hyper-plane in the input space, so the number of required neurons cannot be fewer than the number of *distinct* inequalities involved in the criterion. For the example criterion, at least three neurons are required to represent the three *distinct* inequalities involved. Accordingly, the parameters of Figure 3.10 can be assigned as follows (not unique):

$$\begin{cases} w_{11} = 5, & w_{21} = -1, & \theta_{h1} = 3 \\ w_{12} = -2, & w_{22} = 4, & \theta_{h2} = 3 \\ w_{13} = 2, & w_{23} = 2, & \theta_{h3} = 3 \end{cases}$$

For the output layer, the criterion to realize is the Boolean formula $h_1 \vee (h_2 \wedge h_3)$. The last step of the mapping procedure is to transform a logic formula into a linear inequality with binary variables. In this example, one possible assignments is $(w_1, w_2, w_3, \theta_{out}) = (6, 4, 2, 5)$.

3.3. Level-based neuromorphic computation

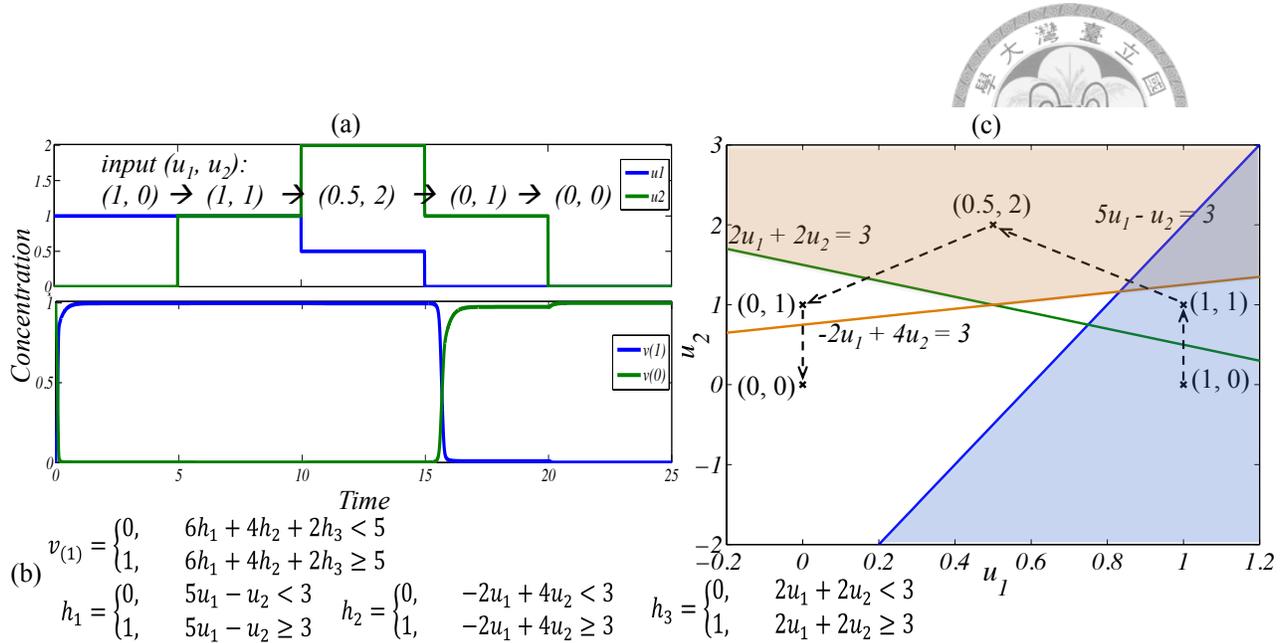


Figure 3.12: (a) Simulation result. (b) Functions implemented by each neuron. (c) Input space and separating hyperplanes. The union of the shaded areas defines the set of inputs with output 1; the arrows indicate the trace of (u_1, u_2) in the simulation.

Figure 3.12(a) shows the *Biocham* simulation result of the corresponding CRN; Figure 3.12(b) summarizes the inequalities implemented by each neuron; Figure 3.12(c) plots the partition of input space given the classification constraints.

Finally, despite the fact that a single hidden layer is enough for any *arbitrary* classification, we would like to end this case study by a brief discussion on cases with multiple hidden layers. For the hidden layers following the first one, each neuron defines its separating function by applying *conjunction* or *disjunction* on the partitions presented by its previous layer, consequently, the transformations are not necessarily linear. This opens up the possibility to directly map some more abstract classification criterion to a specific node. The nested logic operations realized in this way can introduce exponential reduction in the total number of neurons required. We refer the readers to [81] for a comprehensive survey on this topic.



3.3.7 Case study: classifier synthesis with learning ability

In this case study, we would justify our claim that autonomous learning ability can be embedded into the proposed biochemical reaction-based module by realizing autonomous weight update. Note that the “supervised” is not in conflict with being *autonomous*, it only means that the correct output corresponding to some given input-(vector) is *available* to the system during training. The proposed system is autonomous also in the sense that even the *correct output* does not need to be provided artificially, but can be derived from signals sensed from the environment.

In real-world application, the input vector and *correct* classification result can both be time-series data of species concentrations read from the environment. For example, the input vector can be the concentrations of a set of potential identified indicators for diabetes, and the *correct answer* corresponds to recent statistics of blood glucose value (which can be obtained by cascading a reaction-based, constant-leakage integrator with a neuron whose threshold equals the upper-bound of normal value). The system can then be trained into diabetes diagnostic or warning device based on those indicators.

For clarity, we demonstrate autonomous adaptation by using the *perceptron learning algorithm* [66] to train the composing neuron into a one-dimensional classifier *on positive real* that outputs 0 when the input is smaller than 6, and outputs 1 otherwise. The training pairs of input and its corresponding correct answer are presented as concurrent concentrations to the neuron with the network structure shown in Figure 3.13. The threshold value θ of the neuron is *arbitrarily* initialized to 3 and remains fixed; the training target is the input weight represented by its positive and negative components w_p, w_n . Let the input weight be initialized to 2, i.e., $(w_p, w_n) = (2, 0)$. Given our goal, the target training result \widehat{w}_p without changing

3.3. Level-based neuromorphic computation

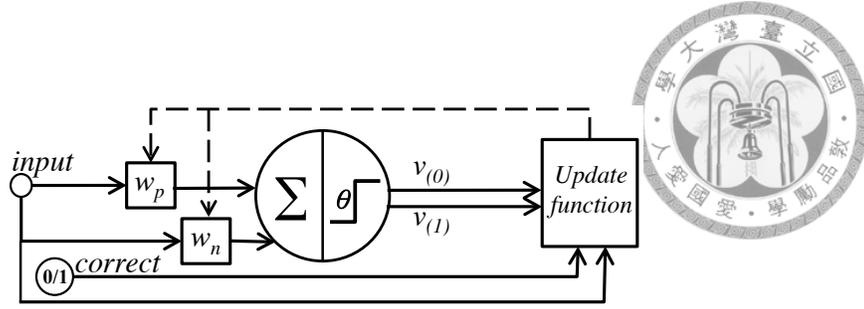


Figure 3.13: Structure of the one dimensional classifier to be trained.

θ is one that satisfies: $(\widehat{w}_p \times input > 3) \equiv (input > 6)$. Hence, our target result is $(\widehat{w}_p, \widehat{w}_n) = (0.5, 0)$. The system keeps comparing its current response with the correct output *continuously in time* as inputs of the training set are fed serially into the system, and updating the weights according to the formula:

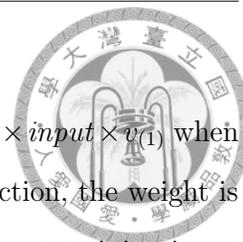
$$w_{i+1} = w_i + \alpha \times input \times (out_{correct} - out_{real}),$$

where w_i is the updated weight after the i^{th} training input is fed. The positive α determines the *learning rate* of the system. The impact of an erroneous output on w_i grows faster under higher learning rate. To implement the update function of the perceptron learning algorithm, the following reactions are added to the neural network CRN.

$$\left\{ \begin{array}{l} input + v_{(1)} \xrightarrow{k_{learn}} input + v_{(1)} + w_n \\ input + correct \xrightarrow{k_{learn}} input + correct + w_p \\ w_n + w_p \xrightarrow{k} \emptyset, \end{array} \right.$$

where the rate constant k here has value similar to the one in neuron implementation, without particular requirement. The reactions work as follows. When the system's output is correct ($v_{(1)} = correct$), w_n and w_p are generated in the same rate by the first two reactions, and the impact will be canceled out by the third reaction. Hence the weight value will not be changed. When $v_{(1)} = 1$ but $correct = 0$, error occurs. The

3.4. Spike-based neuromorphic computation



weight's negative component w_n is produced at rate $K_{learn} = k_{learn} \times input \times v_{(1)}$ when no positive component is produced. Combined with the third reaction, the weight is reduced at constant rate K_{learn} for a time period Δt before the next training input comes in. The weight is thus updated by $(K_{learn} \times \Delta t)$, an increase that conforms with the update rule. Finally, when an error occurs in another direction with $v_{(1)} = 0$ and $correct = 1$, w_p is produced and no negative component is produced in the same period. Following similar reasoning, the weight is updated by $(-k_{learn} \times input \times \Delta t)$, a decrease.

The CRN simulation results of the training process under input series in Figure 3.14(a) are shown. Note that the proposed system allows *online* learning, so can be tuned in real-time as the training inputs come in. Figure 3.14(b) nicely approximates the correct training result with appropriate learning rate, and the module's output value $v_{(1)}$ conforms better with expected output as training proceeds. Figure 3.14(c) and Figure 3.14(d) show the system's behaviors when the learning rate is too large or small, leading to oscillation or slow convergence respectively.

3.4 Spike-based neuromorphic computation

In this subsection, we go one step further toward realizing neuromorphic computation in biochemical reaction systems. Apart from adopting the more realistic spiking model of neuron, considering that it is best to implement the engineered system with *existing* reactions and species in the targeted biological system, we explicitly take the final targeted system into account earlier, at the synthesizing stage.

Cell signaling pathway is thus chosen as the biochemical chassis for our implementation because of its *ubiquity* and *versatility*. Its existence in various types of cells

3.4. Spike-based neuromorphic computation

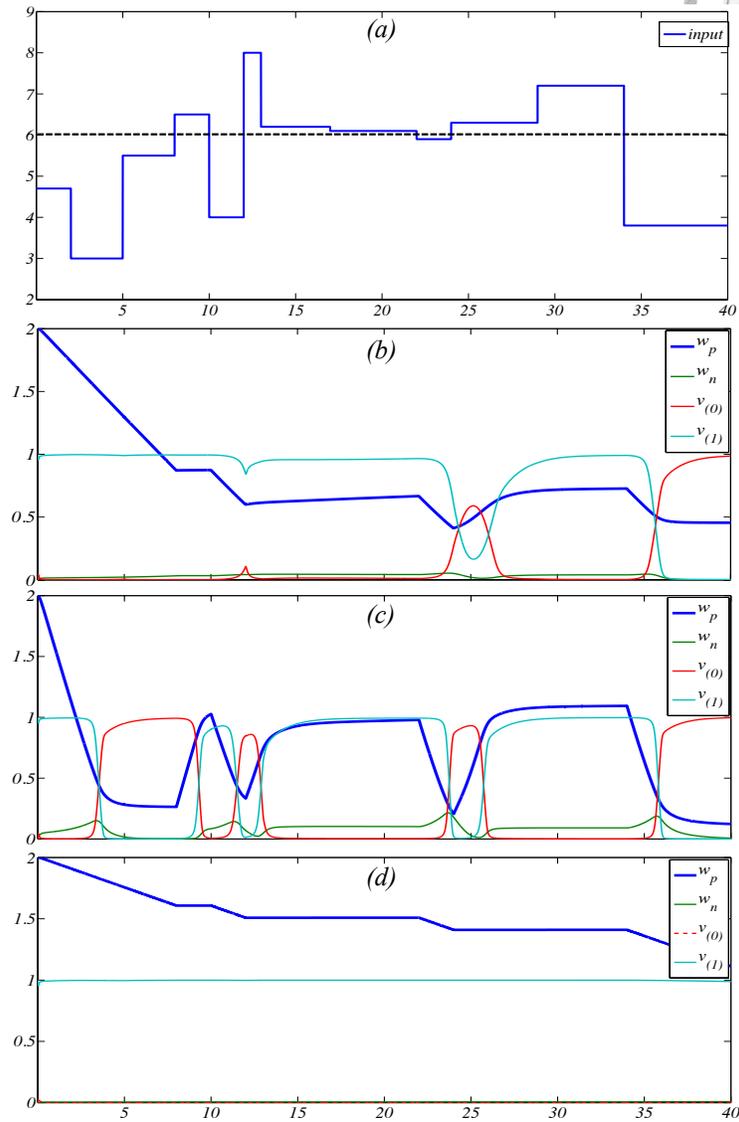
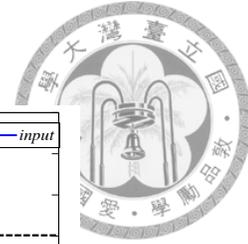
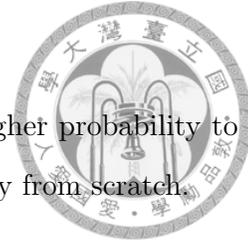


Figure 3.14: (a) Training input series. The applied period of each input pattern can vary, as long as the corresponding *desired output* is presented concurrently. (b)-(d) Simulation results under different learning rates k_{learn} : (b) appropriate, under k , (c) too fast, under $3k$ (d) too slow, under $k/3$.

and involvement in numerous cellular processes [7] makes it an ideal substrate for migrating neuronal functionality to target cells with high compatibility. Besides, the tendency of different biological systems to preserve similar functional modules makes it possible to borrow the wisdom from one system and re-implement it using modules

3.4. Spike-based neuromorphic computation



of similar roles in another system. The resulted system has higher probability to robustly reproduce the desired functions than if crafted manually from scratch.

To sum up, we propose in this section a biocompatible neuromorphic computing system, built with existing reactions in cell signaling pathway. The system also adopts an FPGA-like, module-based architecture that is both self-adaptive and externally-reconfigurable, while each module now corresponds to either a *spiking* neuron or a synapse with plasticity.

3.4.1 Adopted neuron model and key properties

Neurons, put simply, transform complex dynamical inputs into corresponding trains of *action potential* in the form of abrupt voltage spikes. As the amplitude of the output action potential stays roughly the same, the *temporal* profile of spikes must hold a crucial role in encoding stimuli information. Furthermore, as demonstrated in [74], in order to realize useful sensory processing in nature, it is required to perform analog computations at a speed faster than that explainable by an averaging mechanism. Thus the seemingly tenacious effort to take into account not only the average *frequency* of spikes, but also the *timing* of individual spikes as an indispensable carrier of its unique share of information, is more than a desperate pursuit of biological faithfulness. Therefore, it is reasonable for our embedded neuromorphic computation to have plasticity also depend on the timing of spikes.

Here we adopt the *Hodgkin-Huxley* model [41], where the behavior of a neuron depends on the coupling (through membrane potential) of two main types of *voltage-gated* ion channels: the sodium channel Na_v , and the potassium channel K_v . “Voltage-gated” here is used to indicate that the channel’s conductance is dependent on the

3.4. Spike-based neuromorphic computation



membrane potential (i.e. the *voltage*). An unspecified leaking channel is also presented in the model for completeness, however with its *constant* conductance that is *significant lower* than other's, it is of small influence on both the static and dynamic behavior of the membrane potential. An equivalent circuit description of electrical properties across and in immediate vicinity to the membrane is given in Figure 3.4.1. Note that The voltage-gated ion channels are modeled as *variable resistors* to imply the dependency.

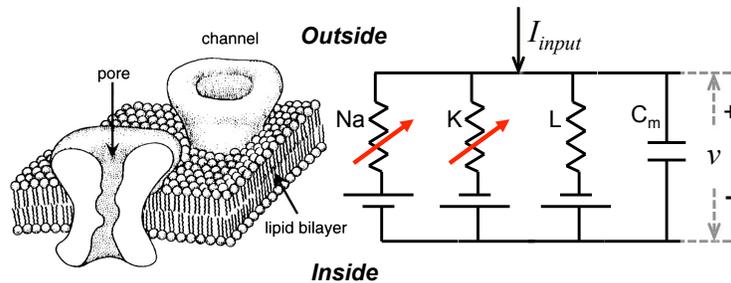


Figure 3.15: The equivalent circuit across the membrane with ion channels' behavior described by the Hodgkin-Huxley model.

While both types of ions are positively charged, the opposite concentration gradient of Na^+ and K^+ across the membrane allows opposite currents (in the form of *ion* movement) to flow *separately* and *selectively* in their respective channels. Depending on the allowed current direction, channels of the same type can either increase or decrease the potential. A simulation of the dynamic evolution of channel current and membrane potential of mutual influence is shown in Figure 3.4.1. The *conductance* of the channels also evolve with the membrane potential. The result of control separation and the tight coupling is a wide range of temporal dynamics adequate for encoding the plentiful input patterns—the response diversity is also one of the crucial requirements while designing our proposed system.



3.4.2 Directional signal transmission with waveform preserved

In biological neural networks, information is encoded and transmitted through neurons' membrane potential. The abilities to preserve the *causality* relation and signal integrity are crucial for the applicability of the knowledge learned.

However, voltage by itself has no directional preference and can potentially spread to all connected neurons including those upstream, masking the correct *causality* relation. The *refractory period* right after each spike as the result of *inactivation* gating mechanism of Na_v and the lag of K_v in closing thus plays a crucial role since it prevents that very spike from re-exciting its source neuron—In the first (*absolute*) part of the refractory period, the neuron that produced the spike cannot fire again no matter how great the stimulation. In the second (*relative*) part, a stronger than usual stimulus is required to trigger the spike. The two periods are distinguished based on whether Na_v has returned from *inactivated* to *close* state. After the refractory period, the neuron will again fire upon reaching the original neural threshold, allowing *directional* propagation of electrical signals in the form of *solitary* waves.

On the other hand, signal integrity concerns the *timing* and the *quality* of the signal—does it reach the destination when it is supposed to? And is the waveform intact upon its reaching? In biological neural network, the shape and velocity of action potential propagation can be kept as nearly constant during axonal propagation between connected neurons, so the information encoded by the source neuron can be well-preserved till reaching the next processing unit. The integrity with well-preserved waveform is achieved in a way similar to how we transfer signals through cables of extended length. The biological counterparts are the cooperation between axon *myelinated* with appropriate thickness and properly distanced *nodes Ranvier*.

3.4. Spike-based neuromorphic computation



3.4.3 Neuron module

The response of biological cells to extra-cellular stimuli is coordinated by networks of protein-based signaling pathways. Signaling pathways can not only *transmit*, but also *process* complex chemical input patterns before encoding the extracted information into signaling activity patterns compatible with targeted down-stream systems. Considering the large variety of control tasks required of the relatively scarce resources, it comes as no surprise that the specificity of diverse physiological signal-response relations is achieved by delicate activation control of the temporal profiles over a restrictive set of signaling proteins, rather than by designating specific, independent pathways to each type of stimulation.

In fact, complex temporal dynamics can arise from modifying reaction *kinetics* and/or *feedback* relations of highly-conserved pathway motif, which captures recurring topological structure across different signaling networks. Figure 3.17 shows (a) the framework of the motif adopted in this paper, and (b) a possible way of forming interconnects between motifs. The motif serves as the backbone structure which, when combined with appropriate feedback design both *in* and *between* modules, can become capable of *signal amplification*, *generation of discontinuous bistable dynamics and oscillations from hysteresis*, etc., enabling the encoding of complex relationships between input stimuli and output cellular responses. More importantly, the *versatility* of the motif is valid with universal applicability—while conserved in organization, what is upstream and downstream can vary widely across species and cells. Systems based on the motif can thus adapt effectively to different types of receptors, substrates and cellular endpoints.

3.4. Spike-based neuromorphic computation

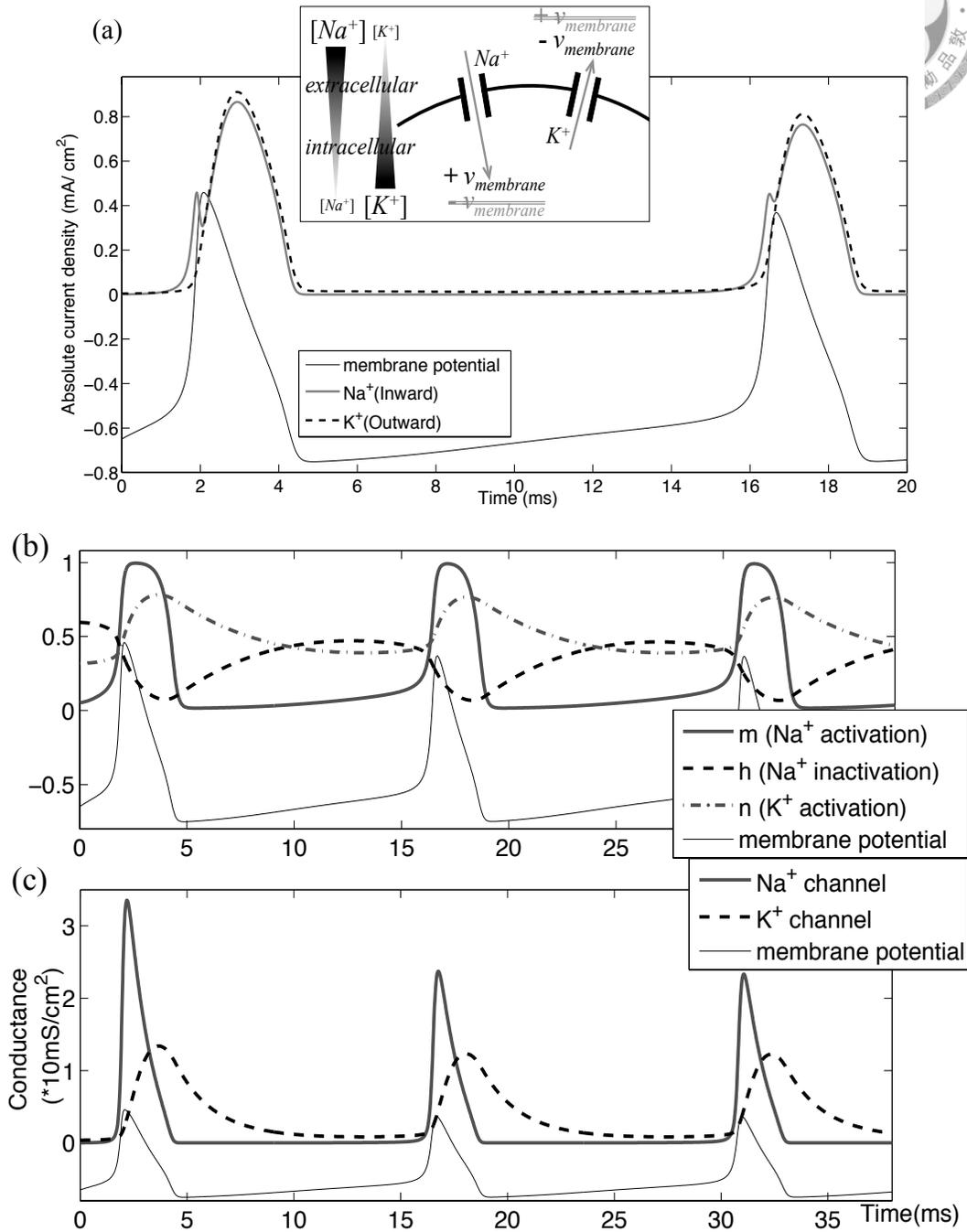


Figure 3.16: A Hodgkin-Huxley neuron's behavior under constant current application: (a) The *absolute* values of current density through ion channels and corresponding membrane potential. *Subfigure:* opposite concentration gradient of Na⁺ and K⁺, and their respective one-way ion flow. (b) Ion activation and deactivation variables' evolution with membrane potential. (c) Membrane potential and the conductance of ion channels. (All membrane potential values are scaled by 0.01 for clarity. Code used for data computation can be found in B.1.)

3.4. Spike-based neuromorphic computation

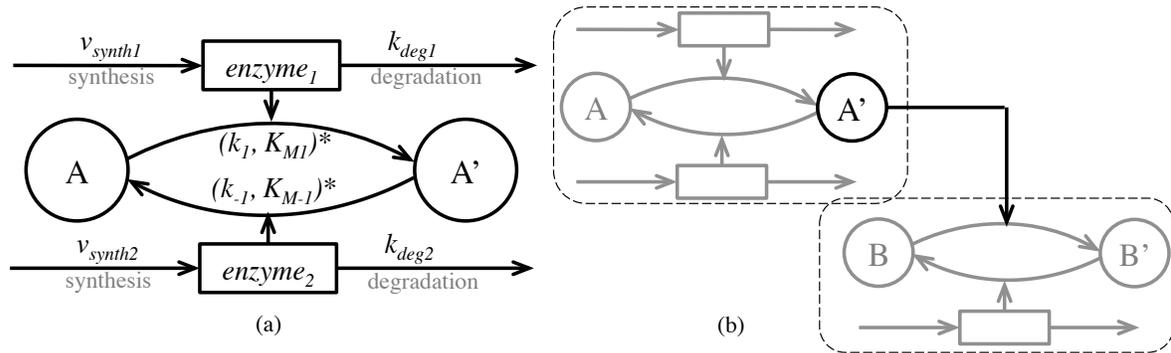


Figure 3.17: (a) The motif consists two forms of a signaling molecule (ex. active-inactive, phosphorylated-dephosphorylated) with constant total amount, mutually interconvertible by different enzymes. The starred enzymatic reactions are assumed to follow Michaelis-Menten kinetics. (b) One possible way of forming an interconnect.



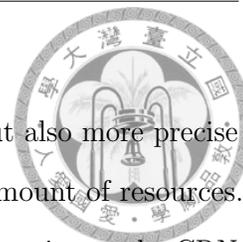
Chapter 4

Technology mapping and implementation

In the previous Chapter 3, different ways of system specification and their corresponding syntheses into biochemical reaction realization described in CRN are presented. In this chapter, the biochemical reaction counterpart of *technology mapping* is presented to realize, or approximate, the interpreted dynamics of CRNs based on the most appropriate semantics for the implementing medium.

One thing to note is that, while in some cases, specifications are synthesized into CRNs without specific assumptions of the final implementing medium, there are also cases when the targeted mapping chassis is itself part of the synthesized specification. To make the best of the information, the synthesized CRNs are then designed based on the reaction structure of the target chassis. More specifically, “motifs,” which can be roughly defined as the *building blocks* of the reaction networks that occur with significantly higher frequency than that would have been if the networks are randomly constructed. The CRNs obtained are thus closer to their final realizations,

4.1. Enzyme



allowing not only easier mapping to existing species/reactions, but also more precise predictions of the resulted systems' behavior and the required amount of resources. However, the advantage comes at the price of reduced flexibility to migrate the CRN design to alternative implementation chassis. It is not uncommon that multiple extra reaction steps are needed to realize, or approximate, the described reactions and kinetics that are designed based on the motifs of another technology.

However, the loss of flexibility serves as a great hint of how combining different implementing technologies into one heterogeneous system may lead to the most accurate and economical realization—which is actually the strategy adopted by living organisms. Life develops through dynamic interactions within metabolic, signaling and gene networks, which are all of different nature, allowing different types of species interactions occurring at rates determined by the species' abundance based on kinetics rules of different forms. Once again, the importance of an efficient hybrid simulator of high accuracy as proposed in Chapter 2 is substantiated.

4.1 Enzyme

Enzymes speed up reactions by providing an alternative reaction pathway of lower activation energy, reducing the energy threshold for a reaction to occur. The most crucial part of an enzyme-mediated reaction involves the binding of the molecule to be transformed (i.e. the *substrate* species) to the enzyme's *active site*. The shape of the active site of an enzyme together with the site's chemical and electrical properties can very effectively prevent the enzyme from reacting with substrates other than the targeted one, endowing enzymes with remarkable chemical specificity. The binding to the active side is crucial for the catalysis process in that the relatively fixed position

4.1. Enzyme



of the substrate allows the barrier-lowering effect to be exerted in various ways, including:

- (a) Enhancing the orientation of reactant molecules.
- (b) Inducing physical strain.
- (c) Inducing chemical changes in the substrates (i.e. reactants) to enhance formation of intermediates.

Two fundamental properties characterize the enzymes from other biochemical species: First, they increase the rate of chemical reactions without themselves being consumed or *permanently* altered by the accelerated reaction, thus the enzymes can repeatedly catalyze a reaction. Second, they increase reaction rates without altering the chemical equilibrium between reactants and products. Therefore, an enzyme simultaneously accelerates both forward and reverse reactions to the same extent.

Enzymes play a dominant role in a wide range of reactions in biological systems due to the fact that: under the mild conditions (ex. temperature and pressure) inside *living* organisms, most biochemical reactions are so slow in the absence of enzymatic catalysis that they can be regarded as being turned *off*. On the other hand, enzymes are able to—very selectively—accelerate the rates of their corresponding sets of reactions by *well over a million-fold*, turning the reactions *on*, allowing the reactions to exert their influence on their hosting biological systems. Consequently, enzymes can effectively take on the much-in-need function of an on-off switch.

To deal with the complexity involved in making cells accomplish a *dynamic* list of functions under *fluctuating* parameter values to survive the always changing environment, cells usually contain thousands of different enzymes, whose concentra-

4.1. Enzyme



tions/activities determine which of the many possible chemical reactions actually take place, and at what speed, within the cell.

In the report issued by *The first Enzyme Commission* in 1961, a system for classification of enzymes was devised, which also serves as a basis for systematically assigning code numbers to each enzyme based on its various properties. These code numbers, prefixed by *EC*, are now widely in use and constantly updated with a devoted website: <http://www.chem.qmul.ac.uk/iubmb/>. Based on the type of reaction an enzyme catalyzes, each enzyme can be categorized into one of the six main classes at the highest classification level: *Oxidoreductases*, *Transferases*, *Hydrolases*, *Lyases*, *Isomerases*, and *Ligases*. Considering the high representativeness and the abundance of the six types of enzymes described, it makes sense to treat them as *motifs*—both topologically and functionally. We will base our modules used for enzymatic technology mapping on the general form of these six categories, as it can increase the probability of finding real-world counterparts of the modules.

To begin with, here we give a brief introduction of the six categories with representing examples in biochemistry:

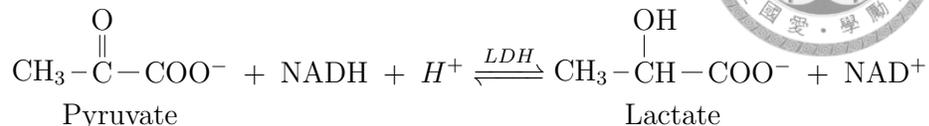
- *Oxidoreductases* catalyze electron-transferring oxidation-reduction reactions. Electrons are transferred from the “reductant” (electron donor) molecule to the “oxidant” (electron acceptor) molecule. In biochemical setting, oxidoreductases, often with NAD(P)H or NAD(P)⁺ as their co-factors, are vital for many metabolic processes, particularly in aerobic and anaerobic respiration.

Lactate dehydrogenase (LDH) is one example found in almost all living cells, catalyzing the interconversion of pyruvate and lactate with NADH being the

4.1. Enzyme

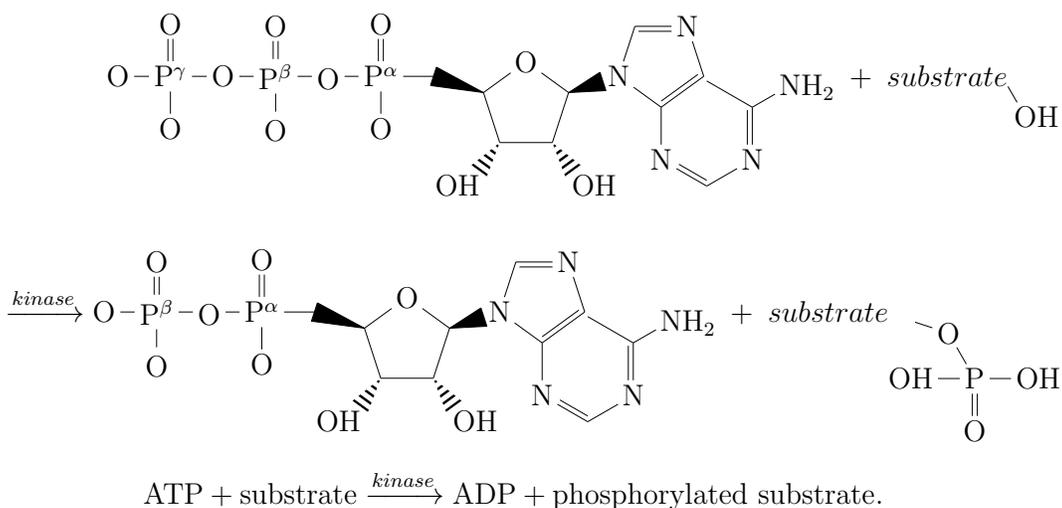


electron donor and NAD^+ the electron acceptor.

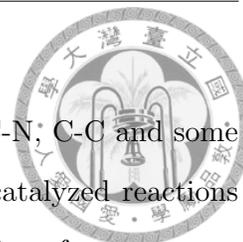


- *Transferases* catalyze functional group transfer reactions. For most cases, the donor of the functional group serves as the coenzyme. Transferases are tightly involved in biological process through their presence in various biochemical pathways. An especially important subgroup is one called the *kinase*.

Kinases catalyze the transfer of phosphate groups in *phosphorylation* process, where the substrate gains a phosphate group from the high energy molecule *ATP* and produces a phosphorylated substrate and ADP. The activity, reactivity, as well as the binding ability of the substrates (including lipids, carbohydrates and nucleotides) are affected by their phosphorylation state. In fact, kinases are critical in metabolism, protein regulation, cell signalling, cellular transport, and many other cellular pathways. The basic phosphorylation reaction has the form:



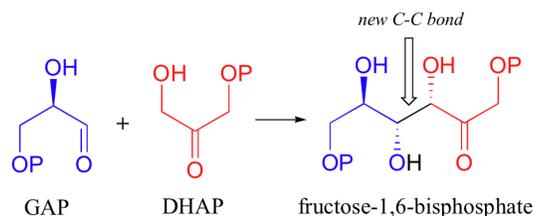
4.1. Enzyme



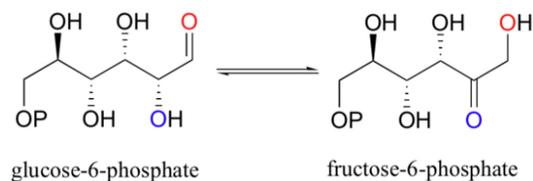
- An *Hydrolase* catalyzes the hydrolysis cleavage of C-O, C-N, C-C and some other bonds such as phosphoric anhydride bonds. The catalyzed reactions involve the breaking of the single bonds through the addition of water:



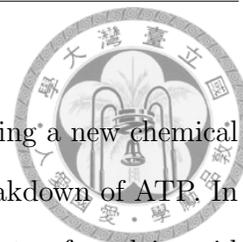
- *Lyases* catalyze non-hydrolytic reactions where functional groups are added with the break of double bonds in molecules, or the reverse where double bonds are formed accompanying the removal of functional groups. *Fructose biphosphate aldolase* serves as one example that converts glyceraldehyde-3-phosphate (GAP) and dihydroxyacetone phosphate (DHAP) to fructose 1,6-bisphosphate:



- *Isomerases* catalyze reactions that transfer functional groups *within* a molecule, so that alternative isomeric forms are produced. The reaction catalyzed has only one substrate yielding one product, both species share the same chemical formula but different chemical structures. The *phosphoglucose isomerase* reaction is one important example in which glucose-6-phosphate (an aldehyde sugar) and fructose-6-phosphate (a ketone sugar) are interconverted.



4.1. Enzyme



- *Ligases* catalyze the joining of two large molecules by forming a new chemical bond (ex. C-O, C-S, C-N or C-C), with simultaneous breakdown of ATP. In biochemistry, ligase can join two complementary fragments of nucleic acid and repair single stranded breaks that arise in double stranded DNA during replication, where *DNA ligase* is the catalyzing enzyme.

Discussions of enzymes as implementing chassis in the rest of this subsection are organized as follows:

In Section 4.1.1, the *Michaelis-Menten kinetics*, one of the most widely adopted enzyme kinetics in biochemistry, is introduced and derived. The model would be used to decide the components' kinetics to guide the selection during technology mapping.

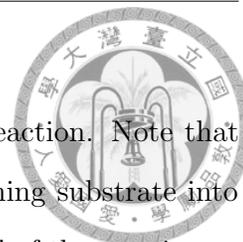
Then in Section 4.1.2, the general form of components based on the six standard categories are presented. Michaelis-Menten kinetics is adopted to model the kinetics realized by each component.

Finally, in Section 4.1.3, the mapping of the configurable logic module proposed in Section 3.2.1 is used as the demonstrating example. The way to find a set of truly existing enzymes that matches the technology-mapped components, with the help of enzyme database and each component's clear correspondence to certain category, is presented.

4.1.1 Enzyme kinetics

The Michaelis-Menten model is a reduction of a catalytic mechanism to a simple 2-step reaction: $E + S \xrightleftharpoons[k_{-1}]{k_1} ES \xrightarrow{k_{cat}} E + P$, where E represents the enzyme, S the

4.1. Enzyme



reactant substrate, and P the final product of the catalyzed reaction. Note that the enzyme is never consumed in the net reaction of transforming substrate into product—it is always regenerated in the original form at the end of the reaction.

Although biochemical reactions involving a *single* substrate are often directly *assumed* to follow Michaelis-Menten kinetics, the derivation of the kinetics actually relies on several simplifying assumptions:

1. In fact, an implicit assumption has already been made to have the product generation step irreversible. The assumption is valid when $[S] \gg [P]$, which is generally true and especially so when the product is continually removed by subsequent reactions.
2. Assuming that the *Mass Action Law kinetics* is applicable.
3. (Quasi-steady-state assumption) The *binding* step ($E + S \xrightleftharpoons[k_{-1}]{k_1} ES$) is fast comparing to the following *catalytic* step ($ES \xrightarrow{k_{cat}} E + P$), so that the $[ES]$ can be regarded as unchanged on the time-scale of product formation. The two assumptions in 2. and 3. lead to the relation: $k_1[E][S] = (k_{-1} + k_{cat})[ES]$. Combining the relation with the enzyme conservation law ¹, the concentration of complex can be derived as $[ES] = \frac{[E]_{total}[S]}{K_M}$, where $K_M \equiv \frac{k_{-1} + k_{cat}}{k_1}$. The assumption is valid when $\frac{[E]_{total}}{[S]_0 + K_M} \ll 1$.
4. $[S] \gg [E_{total}]$, so the fraction of S that binds to E (forming ES) is negligible, and $[S]$ remains near constant throughout the process.

¹ $[E] + [ES] = [E]_{total} = const.$

4.1. Enzyme



Under the assumptions above, the rate v of the reaction can be derived as:

$$v = \frac{d[P]}{dt} = \frac{V_{max}[S]}{K_M + [S]},$$

where $V_{max} = k_{cat}[E]_{total}$ represents the maximum rate achievable by the system under current *total* enzyme concentration (i.e. saturated by substrate).

Figure 4.1 gives a typical reaction rate profile as a function of substrate concentration. Due to the asymptotic behavior at high substrate concentration, the rate is robust to substrate concentration variation at enzyme saturation. And the reaction rate shows *linear* dependency on enzyme concentration at the saturated phase:

$$v \approx V_{max} = k_{cat}[E]_{total} \propto [E]_{total}, \text{ when } [S] \gg K_M.$$

The robustness and predictability make enzyme reactions at their saturated phase an especially attractive building blocks for biomolecular computation implementation.

In fact, Michaelis-Menten model can be regarded as a combination of zero- and first-order kinetics under different substrate concentration relative to the amount of enzyme. Roughly speaking, when substrate concentration $[S]$ is relatively high (i.e. saturated enzyme reaction), the rate equation is zero-order in $[S]$; when $[S]$ is low, Michaelis-Menten equation can be approximated as first order in $[S]$:

$$v \approx \frac{k_{cat}}{K_M}[E]_{total}[S], \text{ when } [S] \ll K_M.$$

4.1. Enzyme

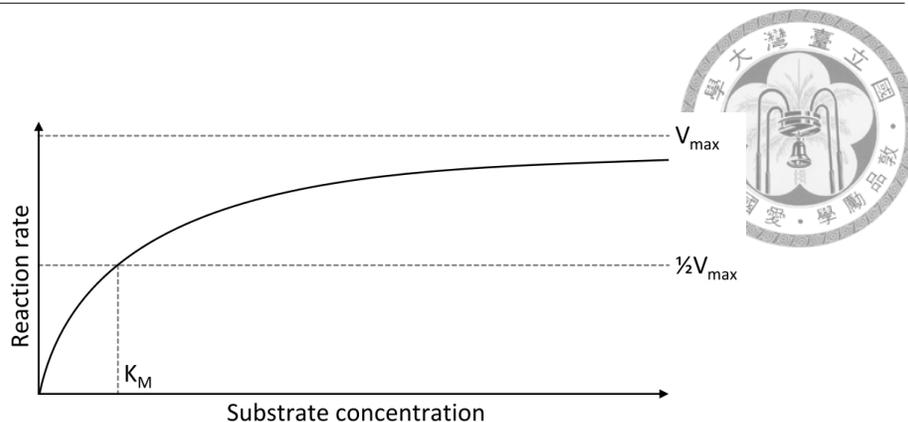


Figure 4.1: Saturation curve for an enzyme reaction showing the relation between the substrate concentration and reaction rate under fixed (total) enzyme concentration.

4.1.2 Reaction motifs

In the framework of systems biology, the search for recurring modules, or *motifs*, usually focuses on *topological* patterns that occur in reaction networks with much higher frequency than expected at random [70]. Novel ways of decomposing networks into topological motifs have kept emerging to deal with the challenges posed by the scale of biochemical networks, further complicated by the intertwined species and reactions prone to variations.

Most works concerning motifs have focused on the *topological* properties. With the kinetics abstracted away, the network can be interpreted with petri-net semantics. However for our purpose, the motifs serve as the bridging medium between the abstract CRNs describing the kinetics that can satisfy the design requirement, and the final realization by existing species and reactions. To interface with kinetics-specifying CRNs, the kinetics of the motifs need to be well-defined, so according to which *kinetics-preserving* technology mapping can be performed and the design requirement remained satisfied. To interface with the implementation chassis, it is preferable to have the motifs *similar* to existing reaction (sub)networks. The similarity requirements should not end at the structural level—different kinetics

4.1. Enzyme



Table 4.1: Enzyme motifs based on the six top-level categories.

Category	Typical reaction (Reaction motif)
EC 1. Oxidoreductases	AH + B → A + BH A + O → AO
EC 2. Transferases	A-group + B → A + B-group <i>Example. cyclin-dependent kinase (CDK):</i> <i>ATP + target protein → ADP + phosphoprotein</i>
EC 3. Hydrolases	AB + H₂O → AOH + BH
EC 4. Lyases	A → B + C <i>Some more detailed common form:</i> <i>RCOCOOH → RCOH + CO₂</i> <i>[X - A - B - Y] → [A = B + X - Y]</i>
EC 5. Isomerases	<i>Structural changes.</i>
EC 6. Ligases	A + B → AB <i>More detailed common form:</i> <i>X + Y + ATP → XY + ADP + Pi</i>

interpretation applied to the same topological structure can easily lead to systems of largely different behavior.

4.1.3 Mapping example: configurable Boolean logic gate

Here we show a possible mapping from the CRN description to components based on the enzyme reaction motifs. The original description and the corresponding enzyme realizations are listed in Table 4.1.3. (We refer the readers to Figure 3.1 in Section 3.2.1 for the system diagram.) In this example, the concentrations of certain predefined *enzymes* are assigned to represent the input values.

The mapping, while mostly direct, involves some twists to take the realistic behavior of enzyme reactions into account. The assumptions made and mapping considerations include:

4.1. Enzyme



Table 4.2: Correspondence between CRN description to enzymatic motifs for realizing reconfigurable logic module proposed in Section 3.2.1. ($\tilde{\emptyset}$: \emptyset or species nonreactive to all listed species or substrates in saturation phase; e^* : enzymes operating in unsaturated phase.)

Synthesized CRN description	Mapped enzymatic motifs
<p>Product of the two inputs (Submodule 1):</p> $\begin{cases} i_1 + i_2 \xrightarrow{k_1} i_1 + i_2 + y_1 \\ y_1 \xrightarrow{k_2} \emptyset \end{cases}$	$\begin{cases} C_1 \xrightarrow{B} D_1 + \tilde{\emptyset} \\ C_2 \xrightarrow{B'} D_2 + \tilde{\emptyset} \\ D_1 + D_2 \xrightarrow{e^*} D + \tilde{\emptyset} \\ B \xrightarrow{e^*_{b}} \tilde{\emptyset} \\ B' \xrightarrow{e^*_{b'}} \tilde{\emptyset} \\ D_1 \xrightarrow{e^*_{d_1}} \tilde{\emptyset} \\ D_2 \xrightarrow{e^*_{d_2}} \tilde{\emptyset} \\ D \xrightarrow{e^*_{d}} \tilde{\emptyset} \end{cases}$ <p>Requirement: $\frac{k[e^*]}{K_M} \ll \frac{k_{d_1}[e^*_{d_1}]}{K_{M_{d_1}}}, \frac{k_{d_2}[e^*_{d_2}]}{K_{M_{d_2}}}$</p>
<p>Sum of the two inputs (Submodule 2):</p> $\begin{cases} a_1 + i_1 \xrightarrow{k_3} a_1 + i_1 + y_2 \\ i_2 \xrightarrow{k_4} i_2 + y_2 \\ y_2 \xrightarrow{k_5} \emptyset \end{cases}$	$\begin{cases} A + BX \xrightarrow{i_1} AX + B \\ A + B'X \xrightarrow{i_2} AX + B' \\ AX \xrightarrow{e^*_{ax}} \tilde{\emptyset} \end{cases}$
<p>Weighted sum from submodule 1 and 2:</p> $\begin{cases} a_2 + y_2 \xrightarrow{k_6} a_2 + y_2 + f_p \\ a_3 + y_1 \xrightarrow{k_7} a_3 + y_1 + f_n \\ a_4 + f_p \xrightarrow{k_8} a_4 \\ f_n \xrightarrow{k_9} \emptyset \end{cases}$	$AX + D \rightarrow \tilde{\emptyset}$ <p>(Exact rate is not a concern as long as it is fast enough.)</p>
<p>Output aggregation:</p> $f_p + f_n \xrightarrow{K} \emptyset$	

4.1. Enzyme



- It is assumed that the uncatalyzed reaction occurrences are very unlikely compared to the catalyzed ones.
- As we can know in advance the possible concentration range of input enzymes, enzymes with appropriate K_M can be selected to satisfy the saturation condition for desired kinetics approximation.
- *Enzyme* is chosen to represent input due to the fact that it is not consumed during reaction occurrences, so connecting the module to an upstream source module does not modify the equilibrium concentration.

Using enzymes as wiring species actually has another advantage that, if the downstream module is designed to operate at *saturation*, it actually helps justify the first assumption introduced in Section 4.1.1 by transforming most enzymes into the form of enzyme-substrate complex.

- Among the reactions synthesized without targeting specific implementing chassis, it is possible to have more than one reactions “reading” the input values, i.e., with the input species as a *reactant* or a catalyzing *enzyme*. When inputs are represented by *enzymes*, this kind of input sharing is relatively unrealistic considering the *specificity* of enzymes. Even when such enzyme exists, the competition for enzyme-binding reduces the *effective* enzyme concentration, thus also the input value truly received by the modules downstream [59].

The problem also occurs in this mapping example, in that i_1 and i_2 are inputs for both submodule 1 and 2. Since the final result is only correct when both submodules operate on the *same* pair of inputs, we solve this competition problem by transforming the parallel CRNs specification into sequential enzyme reactions.

4.1. Enzyme



Auxiliary species

As mentioned in Section 3.2.1, the main purpose of adding auxiliary species is to lift the exact rate constant matching requirement. The more common interpretation is through the product term in the law of mass action—where there is no real distinction between the rate constant k and the concentration of a reactant in their roles in determining reaction rate. The rate constant matching problem can be transformed into one of auxiliary species concentration control.

In enzyme-based implementations, the kinetics behavior of *non-competitive inhibitors* enables direct reaction rate tuning by the inhibitor's concentration [19, 78], without evoking extra mass action interpretation that may not be always applicable. Non-competitive inhibition is very common with multisubstrate enzymes. A non-competitive inhibitor binds to the enzyme at a site *distinct* from the substrate binding site, so the inhibitor I is able to bind to both the free enzyme E and the enzymesubstrate complex ES . The inhibition effect is therefore not through the competition for enzyme binding site, but by preventing complex with I from producing final product. For example, as shown in Figure 4.2, neither EI nor ESI can form final product.

Mathematically, noncompetitive inhibitors directly apply their influence through decreasing the maximum reaction rate without changing the value of K_M . The rate is given by:

$$v = \frac{V_{max}}{K_M + [S]} \frac{[S]}{\left(1 + \frac{[I]}{K_I}\right)}, \text{ i.e. the effective } V_{max} \text{ is scaled by a factor of } \left(1 + \frac{[I]}{K_I}\right)^{-1}.$$

As a result, the concentration of the inhibitor can be used directly to control the

4.1. Enzyme

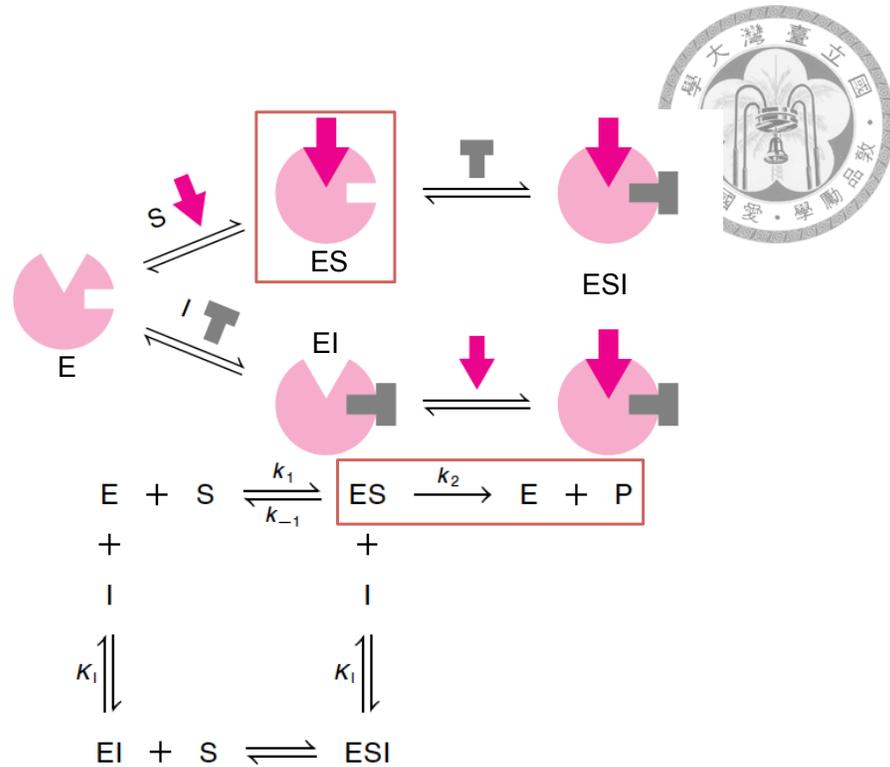


Figure 4.2: Noncompetitive Inhibition. The inhibitor binds to a site other than the active site. Only parts with red framing contribute to the formation of final product. The ESI complex on the very right does not lead to product formation. (Figure adapted from: [16]).

reaction rate at saturation.

Searching for compatible existing enzymes

After mapping to enzymatic components with more general specifications, a comprehensive enzyme database with information on enzyme functions, related organisms, and the values of functional parameters such as (K_M, k_{cat}) in MichaelisMenten kinetics, turnover number, specificity, and functioning environmental range, etc., can be of great help in finding the real-world mapping and in making another important step toward feasibility.

BRENDA [15,69] is the main collection of enzyme functional data available to the

4.2. DNA strand displacement



scientific community. Accessible free of charge via the internet (www.brenda-enzymes.org), the database systematically provides not only the information listed above, but also the referenced sources of the information. With the help of BRENDA, we are able to justify our technology mapping result by simulating with the parameters of truly existing enzymes. Even when no appropriate real-world mapping can be found, the systematic screening made possible by the dataset allows early modification of the mapping result for a better chance to be biologically realizable.

4.2 DNA strand displacement

Nucleic acids play a dual role in biology: their sequences and expression levels together determine the state of a cell, while regulatory RNAs can actively influence the state according to the instructions stored in the very same sequences. The coupled capabilities of *encoding* the desired behavior and actually *realizing* the encoded functions make nucleic acids an ideal substrate for technology mapping from the synthesized CRN, which also encode both the *state variables* of the system in *species*, and how the state should evolve as a function of those variables' values in *reactions*.

DNA's reactions based on their Watson-Crick binding thermodynamics are proved to be an optimal mapping target with their programmability through sequence design. Given a set of DNA strands, all possible interactions can be fully determined solely by the strands' *sequences*, the nucleic sequence itself thus can act as the target to which the synthesized reactions can be mapped in an orderly way once the correspondence is established.

However, while state transitions specified by CRNs are also expected to proceed in

4.2. DNA strand displacement



rate well-defined by the accepted law of mass action in order to achieve the required behavior, it is impossible to realize systematic mapping to an implementation chassis that correctly fulfills the requirement if we cannot find the way to reaction rate control.

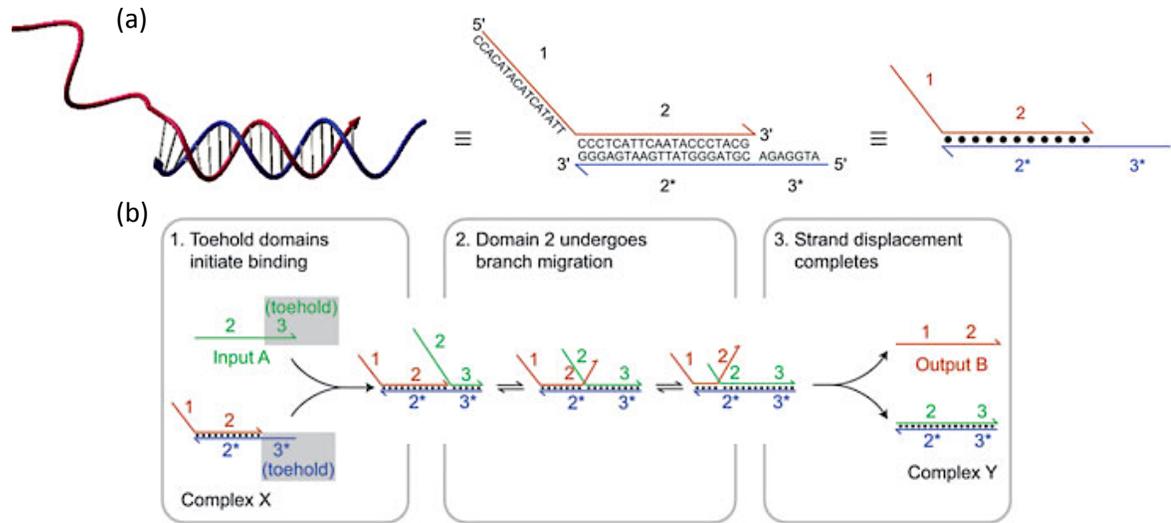
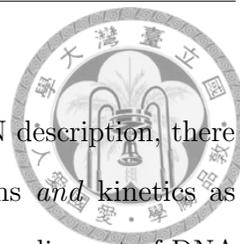


Figure 4.3: (a) Contiguous DNA bases are abstracted into functional DNA domains acting as units in *hybridization*, *branch migration* or *dissociation*. Domains are recognized by numbers; a starred domain denotes the one with complementary sequence to that without a star. (b) An example DNA strand displacement reaction. (Figure adopted from Box 1 in [79])

The *DNA strand displacement (DSD) reaction* [72,79] successfully deals with the concerns above, thus is adopted as one experimental chassis for the final mapping step in our design flow. As illustrated in Figure 4.3(b), one strand displacement involves a strand of DNA displaces another in binding to a third strand of partial complementarity to both. In the example, the hybridization of the single stranded “toehold” domains 3 and 3* initialize the reaction by allowing *branch migration* through domain 2. Branch migration is the random walk process in which one domain displaces another of identical sequence through a series of reversible single nucleotide dissociation and hybridization steps [64].

4.2. DNA strand displacement



It has already been shown in [72] that given an arbitrary CRN description, there always *exists* a DSD reaction realization of the transformations *and* kinetics as described—specifically, each CRN can be emulated by its corresponding set of DNA strand displacement gates [53]. One thing to take notice is that a single-step reaction in the original description usually requires a sequence of multiple reactions by the gates to emulate. Take the *two-domain* encoding schemes adopted in this section as an example, attractive as they were since only simple strands and gates without overhangs are used, additional intermediate steps like garbage collection that converts leftover species into unreactive wastes to prevent them from slowing down certain reactions are required. In the following discussion of this section, the “*gates*” all refer to the “*two-domain DNA strand displacement gates*,” which react with the restricted class of two-domain single strands signals consisting of one toehold domain and one recognition domain.

Apart from the universal existence of such a behavior-preserving mapping, other important properties that make DSD reaction an optimal technology mapping target for rational system design are exactly its *programmability* through sequence design and its highly sequence-dependent, thus *predictable* interactions and kinetics.

The predictability is the combined result of several properties innate to the nucleic molecules. First, a *single* base mismatch is enough to significantly impede branch migration, leading to better system robustness to undesirable reactions. Besides, as we focus ourselves on *toehold-mediated* strand displacement, the reaction rate constant can be modulated over 6 orders of magnitude simply by varying the binding strength of the initiating toeholds, which can be controlled by the length and sequence composition of the toeholds.

To reduce redundant illustrations, in Section 4.2.3 and 4.2.4, we would directly

4.2. DNA strand displacement



continue with the examples from previous sections (3.2.5 and 3.3.6) with their synthesized CRN descriptions, and show the mapping results as their corresponding sets of DNA strands and concentrations. All simulations are conducted using *Visual DSD*; programs are written in a textual syntax described in [54], which supports modules and local parameters to allow for abstraction and code-reuse.

Visual DSD can be downloaded at <http://research.microsoft.com/en-us/projects/dna/>. An online version is accessible at <http://boson.research.microsoft.com/webdna/>.

4.2.1 DSD reaction modules as mapping target

4.2.2 Reaction rate

Despite the complexity of the underlying mechanism, for a wide range of experimental conditions, toehold binding is *rate limiting* and toehold-mediated strand displacement can be well-modeled by a three-step, reversible reaction [80]. It is also shown in the same paper that the rate of dsd reactions can be determined by the strands involved with high accuracy, which serves as the basis of exact strand assignment in *Visual DSD* given desired kinetics requirements.

4.2.3 Mapping example: circuit

Here we show the mapping from CRN to DSD strands of the circuit example described in Section 3.2.5.

Note that the signals are all represented by single-stranded DNAs.

4.2. DNA strand displacement

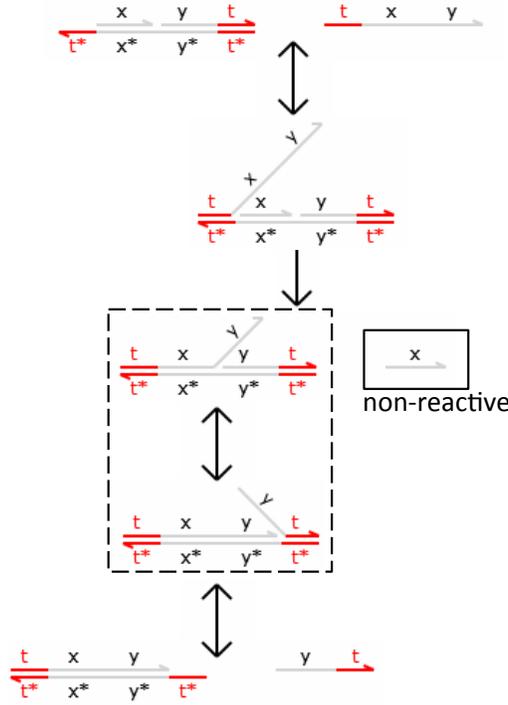


Figure 4.4: The three-step, reversible model of a toehold-mediated DSD reaction.

The simulation results by *Visual DSD* [55] of four *static* inputs evaluated by two time-multiplexed boolean functions:

$$f(x_1, x_2, x_3, x_4) = f_1(0, 1, 0, 1), f_1(0, 0, 0, 0), f_2(1, 1, 1, 1), f_2(1, 1, 0, 0)$$

are shown in Figure 4.6, 4.7, 4.8, and 4.9, respectively. The complete code used is given in Appendix A.1 for reference and reproduction.

4.2.4 Mapping example: classifier

Here we show the mapping from CRN to DSD strands of the classifier example described in Section 3.3.6.

4.2. DNA strand displacement

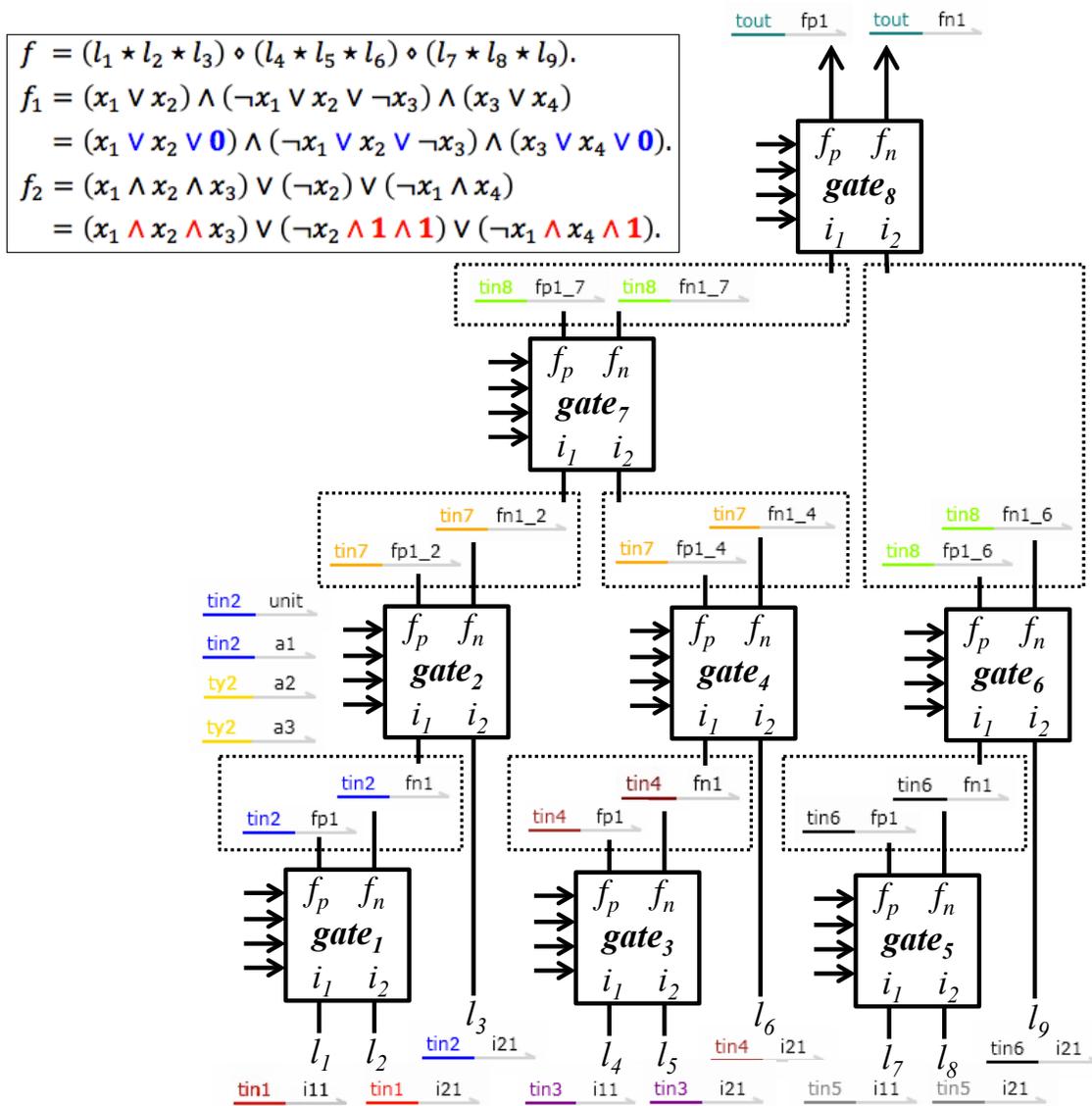


Figure 4.5: The strands and their corresponding signals in the circuit.

The correspondence between the CRN description of the design and the mapping result are given in 5 figures—from Figure 4.11 to Figure 4.15. The simulation results under three *static* inputs $(u_1, u_2) = (1, 0), (0.5, 2)$ and $(0, 0)$ with different classification outputs are shown in Figure 4.10. The complete code used is given in Appendix A.2 for reference and reproduction.

4.2. DNA strand displacement

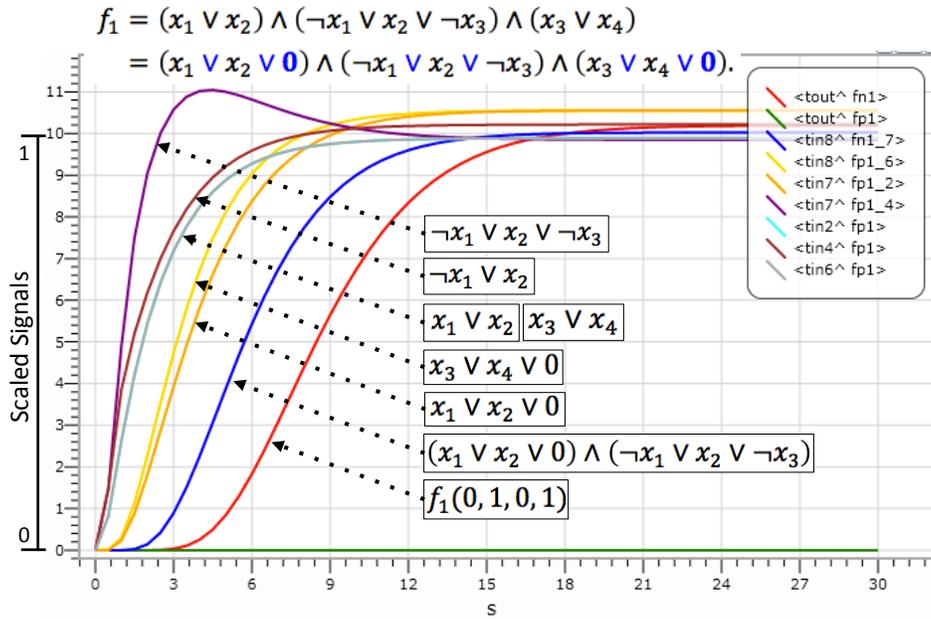
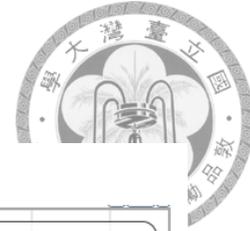


Figure 4.6: DSD *deterministic* simulation result of $f_1(0, 1, 0, 1)$.

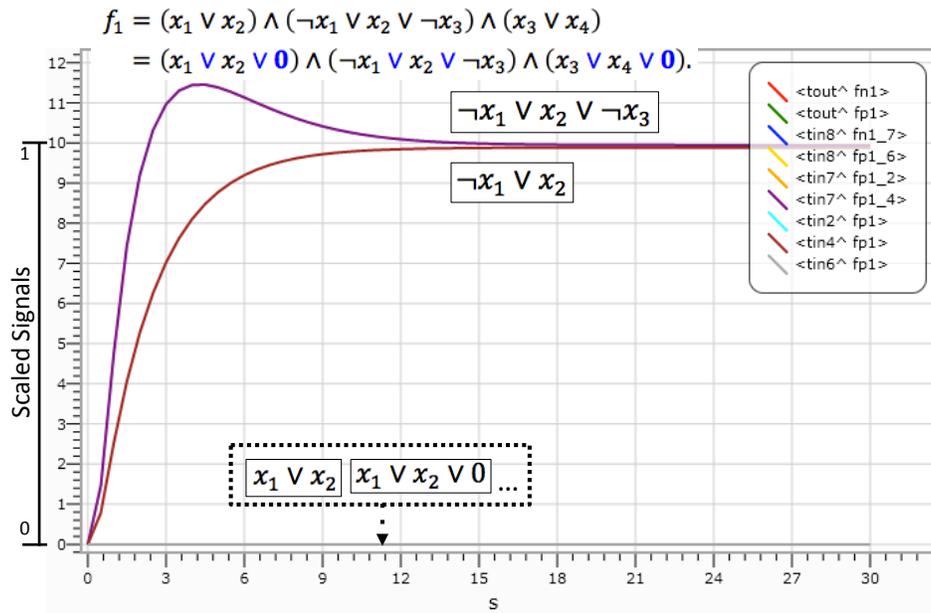


Figure 4.7: DSD *deterministic* simulation result of $f_1(0, 0, 0, 0)$.

4.2. DNA strand displacement

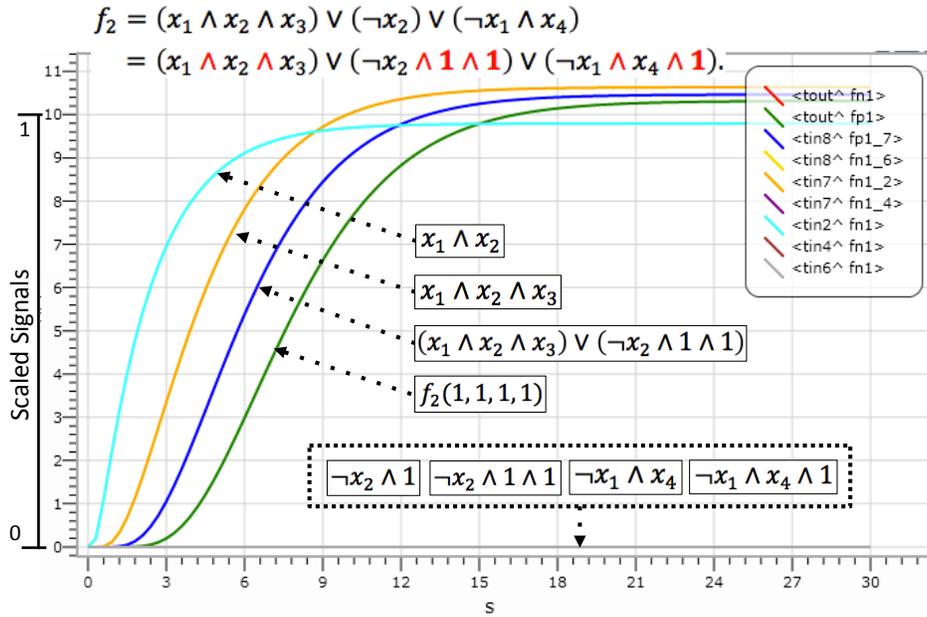
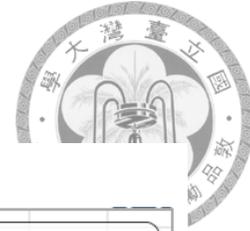


Figure 4.8: DSD *deterministic* simulation result of $f_2(1, 1, 1, 1)$.

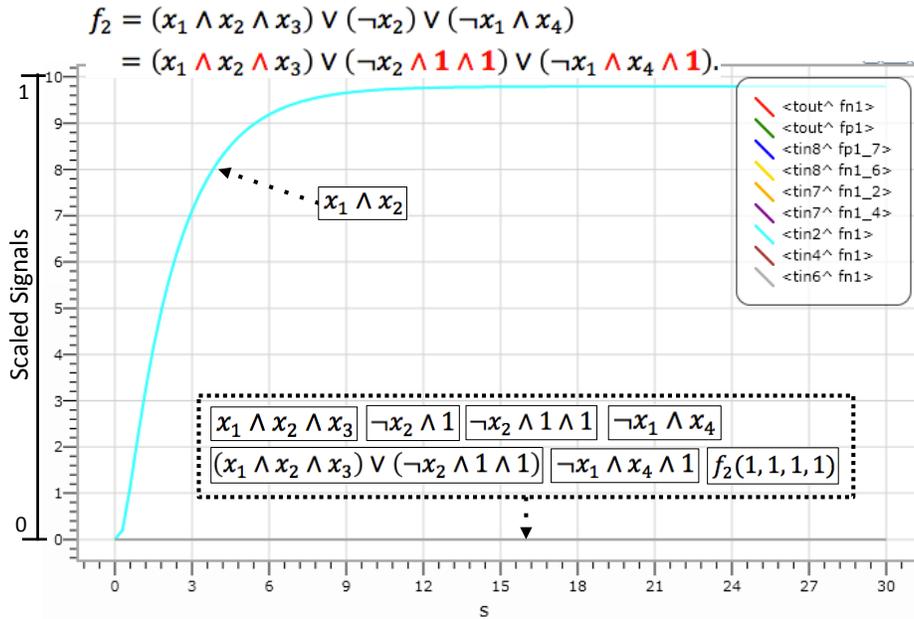


Figure 4.9: DSD *deterministic* simulation result of $f_2(1, 1, 0, 0)$.

4.2. DNA strand displacement

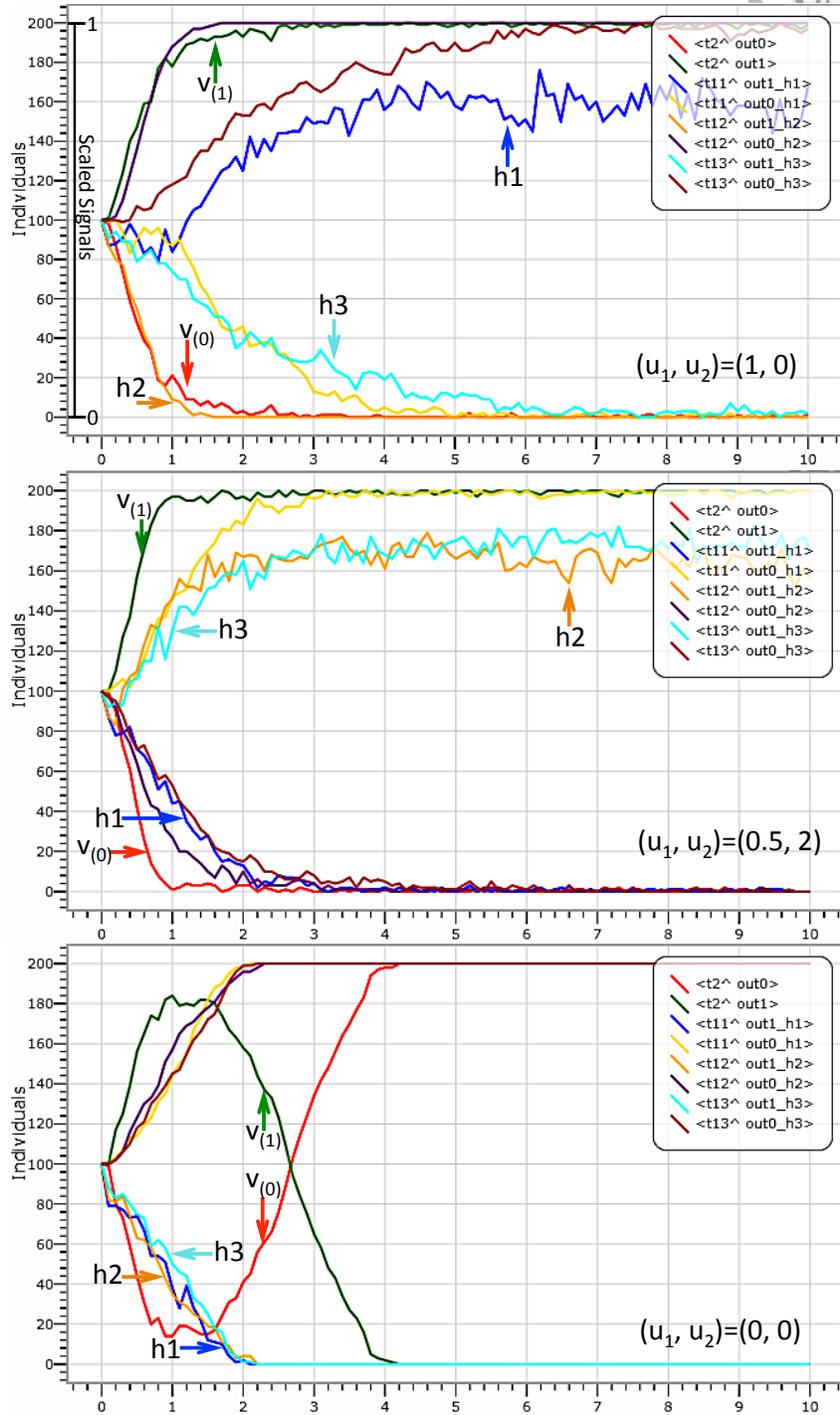


Figure 4.10: DSD stochastic simulation results for classification of static inputs.

4.2. DNA strand displacement

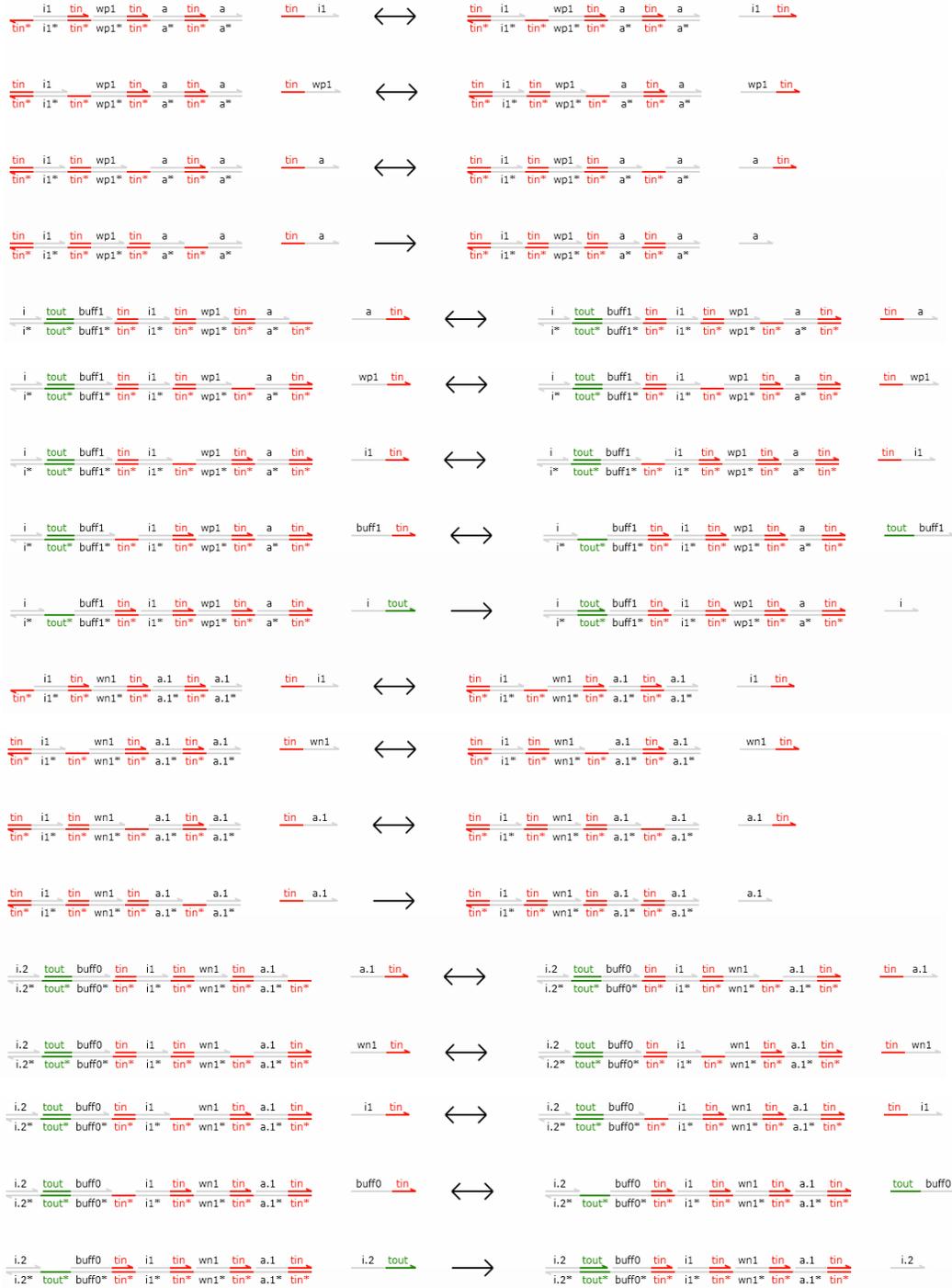
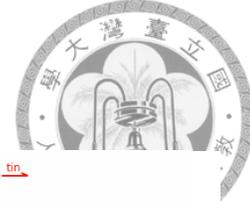


Figure 4.11: DSD reactions for a weighted input of a neuron.

4.2. DNA strand displacement

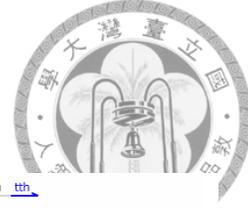


Figure 4.12: DSD reactions for neuron's threshold. Note that it has the same construct as in Figure 4.11 as threshold is interpreted and realized as a (negatively-)weighted input in our design.

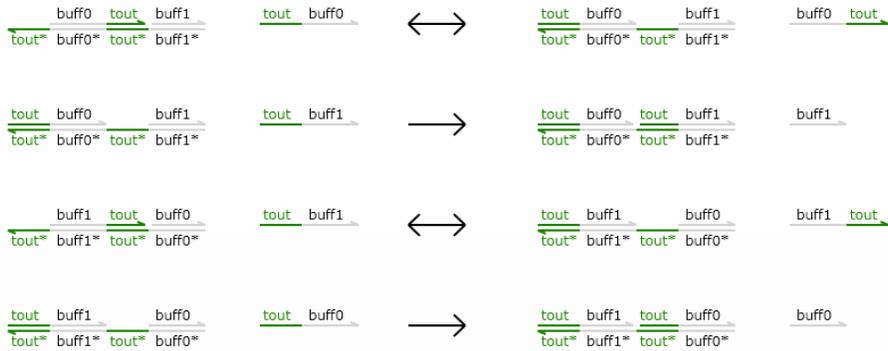


Figure 4.13: DSD reactions that compares inputs' weighted sum with threshold.



Chapter 5

Conclusions and future work

In this thesis, we present a new framework for constructing synthetic biological systems that comes with different formats for specifying system behavior as the input interface. The construction goes all the way through high-level synthesis to technology mapping. Considering biochemical system's generally influential variations, special focus is given to embed reconfigurability into the designed system, so the designed systems' functions can remain as specified under various scenarios.

However, the design verifications are currently done solely through simulations, which is non-exhaustive and cannot be used as a sufficient evidence of correctness. To truly justify the system in order to make the proposed system more attractive to the synthetic biology community and to the wider audience, wet-lab experiments as well as a formal verification paradigm are definitely needed. To realize our *formally verified* designed system, *in vivo* or *in vitro*, would be the most urgent next step of this research.



Appendix A

DSD simulation code

In this appendix, the complete *Visual DSD* codes used for simulations in Section 4.2.3 and 4.2.4 are given. Comments are wrapped between “(*)” and “*),” which briefly describe the functions of corresponding parts of the program.

A.1 Code for mapping example: circuit

Specifically, the code for the case of input $(x_1, x_2, x_3, x_4) = (1, 1, 1, 1)$ evaluated by

$$f_2 = (x_1 \wedge x_2 \wedge x_3) \vee (\neq x_2) \vee (\neg x_1 \wedge x_4)$$

```
.  
  
1 (*=====*)  
2 Implementing (x1)(x2)(x3)+(-x2)+(-x1+x4)  
3 (*=====*)  
4 directive sample 30.0 60  
5 directive plot  
6 <tout^ fn1>; <tout^ fp1>;
```

A.1. Code for mapping example: circuit



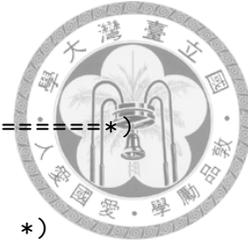
```
7 <tin8^ fp1_7>; <tin8^ fn1_6>;
8 <tin7^ fn1_2>; <tin7^ fn1_4>;
9 <tin2^ fn1>; <tin4^ fn1>; <tin6^ fn1>
10
11 (* different toeholds to avoid loading effect *)
12 new tin1@1.0, 1.0
13 new tin2@1.0, 1.0
14 new tin3@1.0, 1.0
15 new tin4@1.0, 1.0
16 new tin5@1.0, 1.0
17 new tin6@1.0, 1.0
18 new tin7@1.0, 1.0
19 new tin8@1.0, 1.0
20 new ty1@1.0, 1.0
21 new ty2@1.0, 1.0
22 new ty3@1.0, 1.0
23 new ty4@1.0, 1.0
24 new ty5@1.0, 1.0
25 new ty6@1.0, 1.0
26 new ty7@1.0, 1.0
27 new ty8@1.0, 1.0
28 new tout@1.0, 1.0
29
30 (*=====
31 definition of basic reactions
32 =====*)
33 (* X + Y -> X + Y + Z *)
34 def catalyst2(N, tin, tout, x, y, z) = new a new i
35   ( constant N * {tin^*}[x tin^]:[y tin^]:[a tin^]:[a]
36     | constant N * [i]:[tout^z]:[tin^x]:[tin^y]:[tin^a]{tin^*}
37     | constant N * <tin^ a>
38     | constant N * <i tout^>
39     | constant N * <z tin^>
40     | constant N * <x tin^>
41     | constant N * <y tin^>
42   )
43
44 (* X -> Y + Z *)
45 def fork(N, t, x, y, z) = new a
46   ( constant N * <t^ a>
47     | constant N * <y t^>
48     | constant N * <z t^>
49     | constant N * t^*:[x t^]:[a t^]:[a]
50     | constant N * [x]:[t^ y]:[t^ z]:[t^ a]:t^*
51   )
52
```

A.1. Code for mapping example: circuit



```
53 (* Y -> 0 *)
54 def degradation(N, y, t) =
55   ( constant N * {t^*}[y]
56   )
57
58 (* u+ + u- -> 0 *)
59 def annihilation(N, t, up, um) =
60   ( constant N * {t^*}[up t^]:[um]
61   | constant N * {t^*}[um t^]:[up]
62   )
63
64 (=====
65 definition of primitives
66 =====*)
67 (* reconfigurable gate *)
68 def gate(N,tin,ty,tout,          (*gate count; toeholds*)
69         i11,i12,i21,i22,        (*data inputs*)
70         a1,a2,a3,unit,fp,fn,    (*auxiliary variables*)
71         fp1,fn1,fp2,fn2)=      (*data outputs*)
72
73   (* multiplication*)
74   ( catalyst2(N,tin,ty,i11,i21,y1)
75   | degradation(5.0,y1,ty)
76
77   (* summation *)
78   | catalyst2(N,tin,ty,a1,i12,y2)
79   | catalyst2(N,tin,ty,unit,i22,y2)
80   | degradation(5.0,y2,ty)
81
82   (* linear combination *)
83   | catalyst2(N,ty,tout,a2,y2,fp)
84   | catalyst2(N,ty,tout,a3,y1,fn)
85   | degradation(1.0,fp1,tout)
86   | degradation(1.0,fn1,tout)
87   | degradation(1.0,fp2,tout)
88   | degradation(1.0,fn2,tout)
89
90   (* output aggregation *)
91   | annihilation(N,tout,fp1,fn1)
92   | annihilation(N,tout,fp2,fn2)
93   | fork(N,tout,fp,fp1,fp2)
94   | fork(N,tout,fn,fn1,fn2)
95   )
96
97 (=====
98 main function
```

A.1. Code for mapping example: circuit



```
99 =====*)
100 def scaling = 10.0
101
102 (* To compensate for unavoidable loading effects *)
103 def scaling_linear1 = 5.0
104 def scaling_linear2 = 5.0
105 def scaling_linear2_5 = 10.0
106 def scaling_linear3 = 6.5
107 def scaling_linear4 = 6.5
108
109 (=== primary data inputs ===)
110 (* Here: l1 ~ l9 = 111 011 011 *)
111 ( 1.0 *scaling*<tin1^ i11> | 1.0 *scaling*<tin1^ i12>
112 | 1.0 *scaling*<tin1^ i21> | 1.0 *scaling*<tin1^ i22>
113 | 1.0 *scaling*<tin2^ i21> | 1.0 *scaling*<tin2^ i22>
114 | 0.0 *scaling*<tin3^ i11> | 0.0 *scaling*<tin3^ i12>
115 | 1.0 *scaling*<tin3^ i21> | 1.0 *scaling*<tin3^ i22>
116 | 1.0 *scaling*<tin4^ i21> | 1.0 *scaling*<tin4^ i22>
117 | 0.0 *scaling*<tin5^ i11> | 0.0 *scaling*<tin5^ i12>
118 | 1.0 *scaling*<tin5^ i21> | 1.0 *scaling*<tin5^ i22>
119 | 1.0 *scaling*<tin6^ i21> | 1.0 *scaling*<tin6^ i22>
120
121 (=== primary control (auxiliary) inputs ===)
122 (* AND: (a2,a3)=(0,1); OR: (a2,a3)=(1,1) *)
123
124 (* logic layer 1 *)
125 | constant 0.0 *scaling_linear1*<ty1^ a2>
126 | constant 1.0 *scaling_linear1*<ty1^ a3>
127 | constant 0.0 *scaling_linear2*<ty2^ a2>
128 | constant 1.0 *scaling_linear2*<ty2^ a3>
129 | constant 0.0 *scaling_linear1*<ty3^ a2>
130 | constant 1.0 *scaling_linear1*<ty3^ a3>
131 | constant 0.0 *scaling_linear2*<ty4^ a2>
132 | constant 1.0 *scaling_linear2*<ty4^ a3>
133 | constant 0.0 *scaling_linear1*<ty5^ a2>
134 | constant 1.0 *scaling_linear1*<ty5^ a3>
135 | constant 0.0 *scaling_linear2_5*<ty6^ a2>
136 | constant 1.0 *scaling_linear2_5*<ty6^ a3>
137
138 (* logic layer 2 *)
139 | constant 1.0 *scaling_linear3*<ty7^ a2>
140 | constant 1.0 *scaling_linear3*<ty7^ a3>
141 | constant 1.0 *scaling_linear4*<ty8^ a2>
142 | constant 1.0 *scaling_linear4*<ty8^ a3>
143
144 (=== tuning auxiliary inputs ===)
```

A.2. Code for mapping example: classifier



```
145 | 1.01*scaling * <tin1^ unit>
146 | 1.01*scaling * <tin1^ a1>
147 | 1.01*scaling * <tin2^ unit>
148 | 1.01*scaling * <tin2^ a1>
149 | 1.01*scaling * <tin3^ unit>
150 | 1.01*scaling * <tin3^ a1>
151 | 1.01*scaling * <tin4^ unit>
152 | 1.01*scaling * <tin4^ a1>
153 | 1.01*scaling * <tin5^ unit>
154 | 1.01*scaling * <tin5^ a1>
155 | 1.01*scaling * <tin6^ unit>
156 | 1.01*scaling * <tin6^ a1>
157 | 1.01*scaling * <tin7^ unit>
158 | 1.01*scaling * <tin7^ a1>
159 | 1.01*scaling * <tin8^ unit>
160 | 1.01*scaling * <tin8^ a1>
161
162 (* define connections between gates *)
163 (* logic layer 1 *)
164 | gate(100,tin1,ty1,tin2,i11,i12,i21,i22,
165       a1,a2,a3,unit,fp,fn,fp1,fn1,fp2,fn2)
166 | gate(100,tin2,ty2,tin7,fn1,fn2,i21,i22,
167       a1,a2,a3,unit,fp_2,fn_2,fp1_2,fn1_2,fp2_2,fn2_2)
168 | gate(100,tin3,ty3,tin4,i11,i12,i21,i22,
169       a1,a2,a3,unit,fp,fn,fp1,fn1,fp2,fn2)
170 | gate(100,tin4,ty4,tin7,fn1,fn2,i21,i22,
171       a1,a2,a3,unit,fp_4,fn_4,fp1_4,fn1_4,fp2_4,fn2_4)
172 | gate(100,tin5,ty5,tin6,i11,i12,i21,i22,
173       a1,a2,a3,unit,fp,fn,fp1,fn1,fp2,fn2)
174 | gate(100,tin6,ty6,tin8,fn1,fn2,i21,i22,
175       a1,a2,a3,unit,fp_6,fn_6,fp1_6,fn1_6,fp2_6,fn2_6)
176
177 (* logic layer 2 *)
178 | gate(100,tin7,ty7,tin8,fn1_2,fn2_2,fn1_4,fn2_4,
179       a1,a2,a3,unit,fp_7,fn_7,fp1_7,fn1_7,fp2_7,fn2_7)
180 | gate(100,tin8,ty8,tout,fp1_7,fp2_7,fn1_6,fn2_6,
181       a1,a2,a3,unit,fp,fn,fp1,fn1,fp2,fn2)
182 )
```

A.2 Code for mapping example: classifier

```
1 (* simulation time and number of samples *)
```

A.2. Code for mapping example: classifier



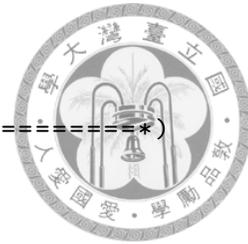
```
2 directive sample 10.0 100
3 (* which strands to plot *)
4 directive plot
5 <t2^ out0>; <t2^ out1>;
6 <t11^ out1_h1>; <t11^ out0_h1>;
7 <t12^ out1_h2>; <t12^ out0_h2>;
8 <t13^ out1_h3>; <t13^ out0_h3>
9
10 (* binding-unbinding rates of toeholds *)
11 new t01@1.0, 1.0
12 new t02@1.0, 1.0
13 new t03@1.0, 1.0
14 new t11@1.0, 1.0
15 new t12@1.0, 1.0
16 new t13@1.0, 1.0
17 new t2@1.0, 1.0
18 new tth1@0.538, 1.0
19 new tth2@0.538, 1.0
20 new tth3@0.538, 1.0
21 new tth@0.577, 1.0
22
23 (*=====
24 mapping of CRN reaction patterns to sets of DSD strands
25 =====*)
26 (* X + Y -> Z *)
27 def join(N, t, x, y, z)= new a new b
28   ( constant N * <t^ a>
29   | constant N * <b t^>
30   | constant N * <z t^>
31   | constant N * t^*:[x t^]:[y t^]:[a t^]:[a]
32   | constant N * [x]:[t^ b]:[t^ z]:[t^ a]:t^*
33   | constant N * t^*:[b y]:t^*
34   )
35
36 (* X + Y -> X + Y + Z *)
37 def catalyst2(N, tin, tout, x, y, z)= new a new i
38   ( constant N * {tin^*}[x tin^]:[y tin^]:[a tin^]:[a]
39   | constant N * [i]:[tout^z]:[tin^x]:[tin^y]:[tin^a]{tin^*}
40   | constant N * <tin^ a>
41   | constant N * <i tout^>
42   | constant N * <z tin^>
43   | constant N * <x tin^>
44   | constant N * <y tin^>
45   )
46
47 (* u+ + u- -> 0 *)
```

A.2. Code for mapping example: classifier



```
48 def annihilation(N, t, up, um)=
49   ( constant N * {t^*}[up t^]:[um]
50   | constant N * {t^*}[um t^]:[up]
51   )
52
53 (*=====
54 define the modules of the system (neuron in this case)
55 =====*)
56 (* 2-input neuron *)
57 def neuron2(N,tin,tout,tth,wp1,wn1,i1,
58            wp2,wn2,i2,ath,th,out1,out0)=
59
60   (* comparison *)
61   ( catalyst2(N, tin, tout, i1, wp1, buff1)
62   | catalyst2(N, tin, tout, i1, wn1, buff0)
63   | catalyst2(N, tin, tout, i2, wp2, buff1)
64   | catalyst2(N, tin, tout, i2, wn2, buff0)
65   | catalyst2(N, tth, tout, ath, th, buff0)
66   | annihilation(10*N, tout, buff0, buff1)
67
68   (* bistable *)
69   | join(10*N, tout, buff0, out1, out0)
70   | join(10*N, tout, buff1, out0, out1)
71   )
72
73 (* 3-input neuron *)
74 def neuron3(N,tin1,tin2,tin3,tout,tth,
75            wp1,wn1,i1,wp2,wn2,i2,wp3,wn3,i3,ath,th,out1,out0)=
76
77   (* comparison *)
78   ( catalyst2(N, tin1, tout, i1, wp1, buff1)
79   | catalyst2(N, tin1, tout, i1, wn1, buff0)
80   | catalyst2(N, tin2, tout, i2, wp2, buff1)
81   | catalyst2(N, tin2, tout, i2, wn2, buff0)
82   | catalyst2(N, tin3, tout, i3, wp3, buff1)
83   | catalyst2(N, tin3, tout, i3, wn3, buff0)
84   | catalyst2(N, tth, tout, ath, th, buff0)
85   | annihilation(10*N, tout, buff0, buff1)
86
87   (* bistable *)
88   | join(10*N, tout, buff0, out1, out0)
89   | join(10*N, tout, buff1, out0, out1)
90   )
91
92 (*=====
93 main function: input and system information
```

A.2. Code for mapping example: classifier



```
94 =====*)
95 (* input (1, 0) *)
96 ( constant 10.0 * <t01^ i1>
97 | constant 0.0 * <t01^ i2>
98 | constant 10.0 * <t02^ i1>
99 | constant 0.0 * <t02^ i2>
100 | constant 10.0 * <t03^ i1>
101 | constant 0.0 * <t03^ i2>
102
103 (* neuron h1: w11=5, w21=-1, threshold=3 *)
104 | constant 50.0 * <t01^ wp1_h1>
105 | constant 0.0 * <t01^ wn1_h1>
106 | constant 0.0 * <t01^ wp2_h1>
107 | constant 10.0 * <t01^ wn2_h1>
108 | constant 10.0 * <tth1^ ath_h1>
109 | constant 30.0 * <tth1^ th_h1>
110 | 100.0 * <t11^ out0_h1>
111 | 100.0 * <t11^ out1_h1>
112
113 (* neuron h2: w12=-2, w22=4, threshold=3 *)
114 | constant 0.0 * <t02^ wp1_h2>
115 | constant 20.0 * <t02^ wn1_h2>
116 | constant 40.0 * <t02^ wp2_h2>
117 | constant 0.0 * <t02^ wn2_h2>
118 | constant 10.0 * <tth2^ ath_h2>
119 | constant 30.0 * <tth2^ th_h2>
120 | 100.0 * <t12^ out0_h2>
121 | 100.0 * <t12^ out1_h2>
122
123 (* neuron h3: w13=2, w23=2, threshold=3 *)
124 | constant 20.0 * <t03^ wp1_h3>
125 | constant 0.0 * <t03^ wn1_h3>
126 | constant 20.0 * <t03^ wp2_h3>
127 | constant 0.0 * <t03^ wn2_h3>
128 | constant 10.0 * <tth3^ ath_h3>
129 | constant 30.0 * <tth3^ th_h3>
130 | 100.0 * <t13^ out0_h3>
131 | 100.0 * <t13^ out1_h3>
132
133 (* neuron out: w1=6, w2=4, w3=2, threshold=5 *)
134 | constant 60.0 * <t11^ wp1>
135 | constant 0.0 * <t11^ wn1>
136 | constant 40.0 * <t12^ wp2>
137 | constant 0.0 * <t12^ wn2>
138 | constant 20.0 * <t13^ wp3>
139 | constant 0.0 * <t13^ wn3>
```

A.2. Code for mapping example: classifier



```
140 | constant 10.0 * <tth^ ath>
141 | constant 50.0 * <tth^ th>
142 | 100.0 * <t2^ out0>
143 | 100.0 * <t2^ out1>
144
145 (* layer 1 *)
146 | neuron2(100,t01,t11,tth1,wp1_h1,wn1_h1,i1,wp2_h1,wn2_h1,i2,
147           ath_h1,th_h1,out1_h1,out0_h1)
148 | neuron2(100,t02,t12,tth2,wp1_h2,wn1_h2,i1,wp2_h2,wn2_h2,i2,
149           ath_h2,th_h2,out1_h2,out0_h2)
150 | neuron2(100,t03,t13,tth3,wp1_h3,wn1_h3,i1,wp2_h3,wn2_h3,i2,
151           ath_h3,th_h3,out1_h3,out0_h3)
152 (* layer 2 *)
153 |neuron3(50,t11,t12,t13,t2,tth,wp1,wn1,out1_h1,
154           wp2,wn2,out1_h2,wp3,wn3,out1_h3,ath,th,out1,out0)
155 )
```



Appendix B

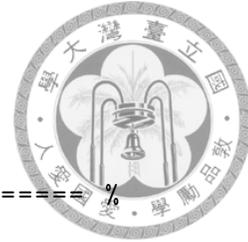
Hodgkin-Huxley model simulation

In this appendix, the complete *Matlab* codes used to demonstrate the combined effect of different ion channels on Hodgkin-Huxley neuron's temporal dynamics are provided.

B.1 Code for mapping example: circuit

```
1 % ===== %
2 % Hodgkin-Huxley model (Keener's Math. Physiology)
3 % (PP.205-206, shifted to physiological equilibrium)
4 %
5 % usage: [t,s]=HH(Iapp,t_end,dt_max,s0)
6 % Inputs:
7 % Iapp - applied current parameter
8 % t_end - ending time
9 % dt_max - maximum time step
10 % s0 - initial condition (V0,m0,h0,n0)
11 % Outputs:
12 % t - time increments
13 % s - phase variables at the time increments
14 % Example inputs:
15 % s0 = [-64.944 0.0533 0.5942 0.3185];(Steady state)
```

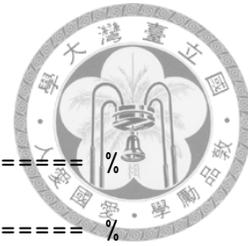
B.1. Code for mapping example: circuit



```
16 % t_end = 100;
17 % dt_max = 0.1;
18 % =====
19 function HH
20 global Iapp
21 s0 = [-64.944 0.0533 0.5942 0.3185];
22 t_end = 100;
23 dt_max = 0.1;
24 tspan=[0 t_end];
25 Iapp=5;
26
27 options=odeset('maxstep',dt_max);
28 [t,s]=ode45(@HH_ODE,tspan,s0,options);
29 % Current density
30 i_Na = (120*(s(:,2).^3).*s(:,3)).*(s(:,1)-56);
31 i_K = (36*(s(:,4).^4)).*(s(:,1)+77);
32 i_L = 0.3*(s(:,1)-10.6+65);
33
34 % === Subfunction for the ODEs === %
35 function s_prime=HH_ODE(t,s)
36 global Iapp
37 V = s(1);
38 m = s(2);
39 h = s(3);
40 n = s(4);
41
42 % Opening and closing rates for gating variables
43 a_m = 0.1*(V+40)/(1 - exp(-(V+40)/10));
44 b_m = 4*exp(-(V+65)/18);
45 a_h = 0.07*exp(-(V+65)/20);
46 b_h = 1/(1+exp(-(V+35)/10));
47 a_n = 0.01*(V+55)/(1-exp(-(V+55)/10));
48 b_n = 0.125*exp(-(V+65)/80);
49
50 % Conductances (mS/cm2)
51 g_K = 36*n^4; % K+ conductance
52 g_L = 0.3; % Leak conductance
53 g_Na = 120*m^3*h; % Na+ conductance
54
55 % The reversal potentials (mV)
56 v_Na = 56; % sodium Nernst potential
57 v_K = -77; % potassium Nernst potential
58 v_L = 10.6-65; % leak equilibrium potential
59
60 % Membrane capacitance (uF/cm2).
61 C_m = 1;
```

B.1. Code for mapping example: circuit

```
62
63 % =====
64 % The derivatives.
65 % =====
66 V_prime = -1/C_m*...
67           (g_Na*(V-v_Na)+g_K*(V-v_K)+g_L*(V-v_L))+Iapp;
68 m_prime = a_m*(1-m)-b_m*m;
69 h_prime = a_h*(1-h)-b_h*h;
70 n_prime = a_n*(1-n)-b_n*n;
71
72 s_prime = [V_prime m_prime h_prime n_prime]';
```





List of Figures

2.1	Comparison of static and dynamic partition strategy with stochastic simulation result. Each curve represent the average of 1000 simulation runs of corresponding strategy, with simulation horizon = 5 time units.	32
2.2	The frequency of each reaction being interpreted as continuous under dynamic partition strategy, over time horizon of 100 units, and calculated over 1000 simulations. Reactions not listed are never interpreted as deterministic during the simulation horizon.	33
2.3	Comparison of post-infection distributions of <i>tem</i> particle counts obtained at time= 50, 100, 150, 200 days, by stochastic, static hybrid and dynamic hybrid simulations (based on 1000 simulation runs of each strategy.)	34
2.4	Evolution of the cyclin's particle counts in Singhania et al. model of the cell cycle. Simulation horizon = 100 hours. (Upper) original continuous-Boolean model (with stochastic delays), with average runtime = 0.15 seconds; (Lower) hybrid stochastic-continuous-Boolean model, with average runtime = 8.42 seconds.	40

List of Figures



3.1 Block diagram of configurable logic unit. 44

3.2 Concentration settings and simulation results of the reconfigurable logic unit. 47

3.3 Retroactivity can be seen as a “reversed” flow from downstream to upstream component. The flow destroys modularity as it actually can create feedback that makes the module’s output(s) iteratively depend on the output’s value, which can lead to unpredictable and/or unstable behavior. For example, the output of central module depends on retroactivities s_p, s_n from downstream modules, but s_p (resp. s_n) is also a function of the module’s output f_p (resp. f_n). The interdependency actually establishes a feedback relation. 51

3.4 (a) A 2-to-1 multiplexer, (b) a parallelized view of its function, and its (c) direct mapping to pass-transistor logic. The reactions listed at the bottom implement the multiplexer function. 52

3.5 A circuit diagram implementing functions f_1 and f_2 . Gates 1 to 6 form the first level of logic; gates 7 and 8 form the second logic level. The four arrows pointing *into* each gate are the *auxiliary inputs*; the dashed rectangles indicate places for wirings: wiring from f_p is established if the source gate implements function *OR*, and f_n in the case of *AND*. 56

3.6 Waveforms of inputs l_1, l_2, \dots, l_9 . The four verticle dashed lines indicate important time points: (1) $t = 10$: connect all wires; (2) $t = 60$: change primary input values; (3) $t = 110$: change implemented function from f_1 to f_2 , and change the value of PIs; (4) $t = 160$: change the value of PIs again. 58

List of Figures



3.7	Waveforms of outputs f_1 and f_2 . Simulation result based on the input values given in Figure 3.6.	59
3.8	A perceptron with three inputs.	62
3.9	A complete digraph network with four perceptrons.	64
3.10	A feed-forward network with one hidden layer.	64
3.11	interconnects of positive and negative weights.	71
3.12	(a) Simulation result. (b) Functions implemented by each neuron. (c) Input space and separating hyperplanes. The union of the shaded areas defines the set of inputs with output 1; the arrows indicate the trace of (u_1, u_2) in the simulation.	74
3.13	Structure of the one dimensional classifier to be trained.	76
3.14	(a) Training input series. The applied period of each input pattern can vary, as long as the corresponding <i>desired output</i> is presented concurrently. (b)-(d) Simulation results under different learning rates k_{learn} : (b) appropriate, under k , (c) too fast, under $3k$ (d) too slow, under $k/3$	78
3.15	The equivalent circuit across the membrane with ion channels' behavior described by the Hodgkin-Huxley model.	80

List of Figures



3.16 A Hodgkin-Huxley neuron's behavior under constant current application: (a) The <i>absolute</i> values of current density through ion channels and corresponding membrane potential. <i>Subfigure</i> : opposite concentration gradient of Na^+ and K^+ , and their respective one-way ion flow. (b) Ion activation and deactivation variables' evolution with membrane potential. (c) Membrane potential and the conductance of ion channels. (All membrane potential values are scaled by 0.01 for clarity. Code used for data computation can be found in B.1.)	83
3.17 (a)The motif consists two forms of a signaling molecule(ex. active-inactive, phosphorylated-dephosphorylated) with constant total amount, mutually interconvertible by different enzymes. The starred enzymatic reactions are assumed to follow Michaelis-Menten kinetics. (b)One possible way of forming an interconnect.	84
4.1 Saturation curve for an enzyme reaction showing the relation between the substrate concentration and reaction rate under fixed (total) enzyme concentration.	94
4.2 Noncompetitive Inhibition. The inhibitor binds to a site other than the active site. Only parts with red framing contribute to the formation of final product. The ESI complex on the very right does not lead to product formation. (Figure adapted from: [16]).	99

List of Figures



4.3	(a) Contiguous DNA bases are abstracted into functional DNA domains acting as units in <i>hybridization</i> , <i>branch migration</i> or <i>dissociation</i> . Domains are recognized by numbers; a starred domain denotes the one with complementary sequence to that without a star. (b) An example DNA strand displacement reaction. (Figure adopted from Box 1 in [79])	101
4.4	The three-step, reversible model of a toehold-mediated DSD reaction.	104
4.5	The strands and their corresponding signals in the circuit.	105
4.6	DSD <i>deterministic</i> simulation result of $f_1(0, 1, 0, 1)$	106
4.7	DSD <i>deterministic</i> simulation result of $f_1(0, 0, 0, 0)$	106
4.8	DSD <i>deterministic</i> simulation result of $f_2(1, 1, 1, 1)$	107
4.9	DSD <i>deterministic</i> simulation result of $f_2(1, 1, 0, 0)$	107
4.10	DSD <i>stochastic</i> simulation results for classification of static inputs. . .	108
4.11	DSD reactions for a weighted input of a neuron.	109
4.12	DSD reactions for neuron's threshold. Note that it has the same construct as in Figure 4.11 as threshold is interpreted and realized as a (negatively-)weighted input in our design.	110
4.13	DSD reactions that compares inputs' weighted sum with threshold. .	110
4.14	DSD reactions for output 0 generation of a neuron.	111
4.15	DSD reactions for output 1 generation of a neuron.	111



List of Tables

2.1	A comparison between purely stochastic and hybrid simulation implemented using chemical reactions and events. Columns <i>#fired events</i> and <i>CPU time</i> respectively hold the number of events triggered and runtime in seconds. All values are the average of 1000 simulations over a time horizon of 100 days. The last row shows the ratio of hybrid to stochastic statistics.	24
2.2	Average number of events fired and average runtime from 100 simulations with simulation horizon set to 100 time units, comparing over three simulation methods. The last two rows are the ratios of dynamic partition strategy's statistics to that of purely stochastic and static partition strategy's, respectively.	32

List of Tables



2.3	Post-infection distributions of <i>tem</i> molecules from simulations using different hybrid strategies compared to the reference fully stochastic model. Each row contains the outcome of applying two-sample Kolmogorov-Smirnov test on the distributions obtained from 1000 simulations using specified hybrid strategy and the reference fully stochastic model. The smaller the value, the more similar the two distributions involved. Distributions at four sampling points are used for comparison through the evolution of time, as listed in four corresponding columns.	35
3.1	Correspondence between circuit signals and function variables.	57
4.1	Enzyme motifs based on the six top-level categories.	95
4.2	Correspondence between CRN description to enzymatic motifs for realizing reconfigurable logic module proposed in Section 3.2.1. ($\tilde{\emptyset}$: \emptyset or species nonreactive to all listed species or substrates in saturation phase; e^* : enzymes operating in unsaturated phase.)	96



Bibliography

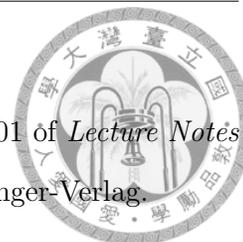
- [1] Jamil Ahmad, Gilles Bernot, Jean-Paul Comet, Didier Lime, and Olivier Roux. Hybrid modelling and dynamical analysis of gene regulatory networks with delays. *ComplexUs*, 3:231–251, 2006.
- [2] Jamil Ahmad, Olivier Roux, Gilles Bernot, Jean-Paul Comet, and Adrien Richard. Analysing formal models of genetic regulatory networks with delays. *International Journal of Bioinformatics Research and Applications*, 4(3):240–262, 2008.
- [3] Ozgur E. Akman, Maria Luisa Guerriero, Laurence Loewe, and Carl Troein. Complementary approaches to understanding the plant circadian clock. In Emanuela Merelli and Paola Quaglia, editors, *Proceedings Third Workshop From Biology To Concurrency and back, FBTC 2010, Paphos, Cyprus, 27th March 2010.*, volume 19 of *EPTCS*, pages 1–19. Open Publishing Association, 2010.
- [4] Aurélien Alfonsi, Eric Cancès, Gabriel Turinici, Barbara di Ventura, and Wilhelm Huisinga. Adaptive simulation of hybrid stochastic and deterministic models for biochemical systems. *ESAIM: Proc.*, 14:1–13, September 2005.
- [5] Rajeev Alur, Calin Belta, Franjo Ivanicic, Vijay Kumar, Max Mintz, George J. Pappas, Harvey Rubin, and Jonathan Schug. Hybrid modeling and simulation of biomolecular networks. In *Proceedings of the 4th International Workshop on*

Bibliography



- Hybrid Systems: Computation and Control, HSCC'01*, volume 2034 of *Lecture Notes in Computer Science*, pages 19–32, Rome, Italy, 2001. Springer-Verlag.
- [6] Michael Ashburner, Catherine A. Ball, Judith A. Blake, David Botstein, Heather Butler, J. Michael Cherry, Allan P. Davis, Kara Dolinski, Selina S. Dwight, Janan T. Eppig, Midori A. Harris, David P. Hill, Laurie Issel-Tarver, Andrew Kasarskis, Suzanna Lewis, John C. Matese, Joel E. Richardson, Martin Ringwald, Gerald M. Rubin, and Gavin Sherlock. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.
- [7] L Bardwell. Mechanisms of mapk signalling specificity. *Biochemical Society Transactions*, 34(Pt 5):837, 2006.
- [8] Martin Bentele and Roland Eils. General stochastic hybrid method for the simulation of chemical reaction processes in cells. In *CMSB'05: Proceedings of the third international conference on Computational Methods in Systems Biology*, volume 3082 of *Lecture Notes in Computer Science*, pages 248–251, Paris, France, 2005. Springer-Verlag.
- [9] Natalie Berestovsky, Wanding Zhou, Deepak Nagrath, and Luay Nakhleh. Modeling integrated cellular machinery using hybrid petri-boolean networks. *PLoS Computational Biology*, 9(11):1003306, November 2013.
- [10] Valeria Bertacco, S Minato, Peter Verplaetse, Luca Benini, and Giovanni De Micheli. Decision diagrams and pass transistor logic synthesis. In *Int'l Workshop on Logic Synth*, volume 168, 1997.
- [11] Alexander Bockmayr and Arnaud Courtois. Using hybrid concurrent constraint programming to model dynamic biological systems. In *Proceedings of ICLP'02*,

Bibliography



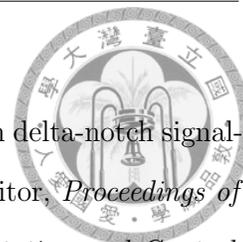
- International Conference on Logic Programming*, volume 2401 of *Lecture Notes in Computer Science*, pages 85–99, Copenhagen, 2002. Springer-Verlag.
- [12] Jennifer A Broderick and Phillip D Zamore. Microrna therapeutics. *Gene therapy*, 18(12):1104–1110, 2011.
- [13] Laurence Calzone, François Fages, and Sylvain Soliman. BIOCHAM: An environment for modeling biological systems and formalizing experimental knowledge. *Bioinformatics*, 22(14):1805–1807, 2006.
- [14] Nathalie Chabrier and François Fages. Symbolic model checking of biochemical networks. In Corrado Priami, editor, *CMSB'03: Proceedings of the first workshop on Computational Methods in Systems Biology*, volume 2602 of *Lecture Notes in Computer Science*, pages 149–162, Rovereto, Italy, March 2003. Springer-Verlag.
- [15] Antje Chang, Ida Schomburg, Sandra Placzek, Lisa Jeske, Marcus Ulbrich, Mei Xiao, Christoph W Sensen, and Dietmar Schomburg. Brenda in 2015: exciting developments in its 25th year of existence. *Nucleic acids research*, page gku1068, 2014.
- [16] Raymond Chang. *Physical chemistry for the biosciences*. University Science Books, 2005.
- [17] Yuan-Jyue Chen, Neil Dalchau, Niranjan Srinivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. Programmable chemical controllers made from dna. *Nature nanotechnology*, 8(10):755–762, 2013.
- [18] H-JK Chiang, J-HR Jiang, and F Fagesy. Building reconfigurable circuitry in a biochemical world. In *Biomedical Circuits and Systems Conference (BioCAS), 2014 IEEE*, pages 560–563. IEEE, 2014.

Bibliography



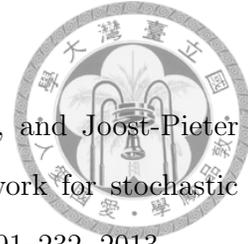
- [19] Athel Cornish-Bowden. A simple graphical method for determining the inhibition constants of mixed, uncompetitive and non-competitive inhibitors. *Biochemical Journal*, 137(1):143, 1974.
- [20] Ramiz Daniel, Jacob R Rubens, Rahul Sarpeshkar, and Timothy K Lu. Synthetic analog computation in living cells. *Nature*, 497(7451):619–623, 2013.
- [21] Domitilla Del Vecchio, Alexander J Ninfa, and Eduardo D Sontag. Modular cell biology: retroactivity and insulation. *Molecular systems biology*, 4(1):161, 2008.
- [22] François Fages, François-Marie Floch, Steven Gay, Dragana Jovanovska, Aurélien Rizk, and Sylvain Soliman. *BIOCHAM v3.6 Reference Manual*. INRIA, 2014.
- [23] François Fages and Sylvain Soliman. Abstract interpretation and types for systems biology. *Theoretical Computer Science*, 403(1):52–70, 2008.
- [24] François Fages and Sylvain Soliman. Formal cell biology in BIOCHAM. In M. Bernardo, P. Degano, and G. Zavattaro, editors, *8th Int. School on Formal Methods for the Design of Computer, Communication and Software Systems: Computational Systems Biology SFM’08*, volume 5016 of *Lecture Notes in Computer Science*, pages 54–80, Bertinoro, Italy, February 2008. Springer-Verlag.
- [25] Marc Frahm, Neela D Goswami, Kouros Owzar, Emily Hecker, Ann Mosher, Emily Cadogan, Payam Nahid, Guido Ferrari, and Jason E Stout. Discriminating between latent and active tuberculosis with multiple biomarker responses. *Tuberculosis*, 91(3):250–256, 2011.
- [26] Vashti Galpin, Jane Hillston, and Luca Bortolussi. {HYPER} applied to the modelling of hybrid biological systems. *Electronic Notes in Theoretical Computer Science*, 218:33–51, 2008. Proceedings of the 24th Conference on the Mathematical Foundations of Programming Semantics (MFPS XXIV).

Bibliography



- [27] Ronojoy Ghosh and Claire Tomlin. Lateral inhibition through delta-notch signaling: A piecewise affine hybrid model. In Springer-Verlag, editor, *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control, HSCC'01*, volume 2034 of *Lecture Notes in Computer Science*, pages 232–246, Rome, Italy, 2001.
- [28] David Gilbert, Monika Heiner, and Sebastian Lehrack. A unifying framework for modelling and analysing biochemical pathways using petri nets. In Muffy Calder and Stephen Gilmore, editors, *CMSB'07: Proceedings of the fifth international conference on Computational Methods in Systems Biology*, volume 4695 of *Lecture Notes in Computer Science*, Edinburgh, Scotland, September 2007. Springer-Verlag.
- [29] Daniel T. Gillespie. General method for numerically simulating stochastic time evolution of coupled chemical-reactions. *Journal of Computational Physics*, 22:403–434, 1976.
- [30] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977.
- [31] Daniel T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *Journal of Chemical Physics*, 115(4):1716–1733, July 2001.
- [32] Daniel T. Gillespie. Deterministic limit of stochastic chemical kinetics. *The Journal of Physical Chemistry B*, 113(6):1640–1644, 2009.
- [33] Andrew Golightly and Darren J. Wilkinson. Bayesian parameter inference for stochastic biochemical network models using particle markov chain monte carlo. *Interface Focus*, 2011.

Bibliography



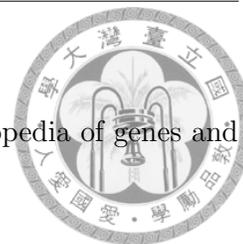
- [34] Ernst Moritz Hahn, Arnd Hartmanns, Holger Hermanns, and Joost-Pieter Katoen. A compositional modelling and analysis framework for stochastic hybrid systems. *Formal Methods in System Design*, 43(2):191–232, 2013.
- [35] Simon Haykin and Neural Network. A comprehensive foundation. *Neural Networks*, 2(2004), 2004.
- [36] Andreas Hellander and Per Lotstedt. Hybrid method for the chemical master equation. *Journal of Computational Physics*, 227(1):100–122, 2007.
- [37] James Hemphill and Alexander Deiters. Dna computation in mammalian cells: microrna logic operations. *Journal of the American Chemical Society*, 135(28):10512–10518, 2013.
- [38] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science (LICS)*, pages 278–292. IEEE Computer Society Press, 1996. An extended version appeared in *Verification of Digital and Hybrid Systems*.
- [39] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A model checker for hybrid systems. In *CAV'97: Proceedings of the 9th International Conference on Computer Aided Verification*, pages 460–463. Springer-Verlag, 1997.
- [40] Thomas A. Henzinger, Linar Mikeev, Maria Mateescu, and Verena Wolf. Hybrid numerical solution of the chemical master equation. In *Proceedings of the 8th International Conference on Computational Methods in Systems Biology, CMSB '10*, pages 55–65, New York, NY, USA, 2010. ACM.

Bibliography



- [41] Alan L Hodgkin and Andrew 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.
- [42] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.
- [43] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.
- [44] De-An Huang, Jie-Hong R Jiang, Ruei-Yang Huang, and Chi-Yun Cheng. Compiling program control flows into biochemical reactions. In *Proceedings of the International Conference on Computer-Aided Design*, pages 361–368. ACM, 2012.
- [45] Michael Hucka et al. The systems biology markup language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003.
- [46] Trey Ideker, Timothy Galitski, and Leroy Hood. A new approach to decoding life: Systems biology. *Annual Review of Genomics and Human Genetics*, 2:343–372, 2001.
- [47] Hua Jiang, Marc D Riedel, and Keshab Parhi. Digital logic with molecular reactions. In *Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference on*, pages 721–727. IEEE, 2013.
- [48] Hua Jiang, Marc D Riedel, and Keshab K Parhi. Digital signal processing with molecular reactions. *IEEE Design and Test of Computers*, 29(3):21–31, 2012.

Bibliography



- [49] Minoru Kanehisa and Susumu Goto. KEGG: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Research*, 28(1):27–30, 2000.
- [50] Jonathan R. Karr, Jayodita C. Sanghvi, Derek N. Macklin, Miriam V. Gutschow, Jared M. Jacobs, Benjamin Bolival Jr, Nacyra Assad-Garcia, John I. Glass, , and Markus W. Covert. A whole-cell computational model predicts phenotype from genotype. *Cell*, 150(2):389,401, 2012.
- [51] Thomas R. Kiehl, Robert M. Mattheyses, , and Melvin K. Simmons. Hybrid simulation of cellular behavior. *Bioinformatics*, 20(3):316–322, 2004.
- [52] M. Kwiatkowska, G. Norman, and D. Parker. Using probabilistic model checking in systems biology. *SIGMETRICS Performance Evaluation Review*, 35(4):14–21, 2008.
- [53] Matthew R Lakin, Andrew Phillips, and Darko Stefanovic. Modular verification of dna strand displacement networks via serializability analysis. In *DNA Computing and Molecular Programming*, pages 133–146. Springer, 2013.
- [54] Matthew R Lakin, Simon Youssef, Luca Cardelli, and Andrew Phillips. Abstractions for dna circuit design. *Journal of The Royal Society Interface*, 9(68):470–486, 2012.
- [55] Matthew R Lakin, Simon Youssef, Filippo Polo, Stephen Emmott, and Andrew Phillips. Visual dsd: a design and analysis tool for dna strand displacement systems. *Bioinformatics*, 27(22):3211–3213, 2011.
- [56] O. Maler and G. Batt. Approximating continuous systems by timed automata. In J. Fisher, editor, *First International Workshop on Formal Methods in Systems Biology, FMSB’08*, volume 5054 of *Lecture Notes in Bioinformatics*, pages 77–89. Springer-Verlag, 2008.

Bibliography



- [57] WR Wayne Martin, Marguerite Wieler, and Myrlene Gee. Midbrain iron content in early parkinson disease a potential biomarker of disease status. *Neurology*, 70(16 Part 2):1411–1417, 2008.
- [58] Hiroshi Matsuno, Atsushi Doi, Masao Nagasaki, and Satoru Miyano. Hybrid petri net representation of gene regulatory network. In *Proceedings of the 5th Pacific Symposium on Biocomputing*, pages 338–349, Hawaii, USA, 2000. Stanford.
- [59] Nomenclature Committee of the International Union of Biochemistry (NC-IUB). Symbolism and terminology in enzyme kinetics: Recommendations 1981. *Archives of Biochemistry and Biophysics*, 224(2):732 – 740, 1983.
- [60] Kazuaki Oishi and Eric Klavins. Biomolecular implementation of linear i/o systems. *Systems Biology, IET*, 5(4):252–260, 2011.
- [61] Jürgen Pahle. Biochemical simulations: stochastic, approximate stochastic and hybrid approaches. *Briefings in Bioinformatics*, 10(1):53–64, 2009.
- [62] Jacek Puchaka and Andrzej M. Kierzek. Bridging the gap between stochastic and deterministic regimes in the kinetic simulations of the biochemical reaction networks. *Biophysical Journal*, 86(3):1357–1372, 2004.
- [63] Lulu Qian and Erik Winfree. Scaling up digital circuit computation with dna strand displacement cascades. *Science*, 332(6034):1196–1201, 2011.
- [64] Charles M. Radding, Kenneth L. Beattie, William K. Holloman, and Roger C. Wiegand. Uptake of homologous single-stranded fragments by superhelical dna: Iv. branch migration. *Journal of Molecular Biology*, 116(4):825 – 839, 1977.
- [65] Raúl Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

Bibliography



- [66] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [67] Howard Salis and Yiannis N. Kaznessis. Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions. *The Journal of Chemical Physics*, 122(5):54103, 2005.
- [68] Howard Salis, Vassilios Sotiropoulos, and Yiannis N. Kaznessis. Multiscale hy3s: Hybrid stochastic simulation for supercomputers. *BMC Bioinformatics*, 7(1):93, 2006.
- [69] Ida Schomburg, Oliver Hofmann, Claudia Baensch, Antje Chang, and Dietmar Schomburg. Enzyme data and metabolic information: Brenda, a resource for research in biology, biochemistry, and medicine. *Gene Function & Disease*, 1(3-4):109–118, 2000.
- [70] Shai S Shen-Orr, Ron Milo, Shmoolik Mangan, and Uri Alon. Network motifs in the transcriptional regulation network of escherichia coli. *Nature genetics*, 31(1):64–68, 2002.
- [71] Rajat Singhania, R. Michael Sramkoski, James W. Jacobberger, and John J. Tyson. A hybrid model of mammalian cell cycle regulation. *PLOS Computational Biology*, 7(2):e1001077, February 2011.
- [72] David Soloveichik, Georg Seelig, and Erik Winfree. Dna as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences*, 107(12):5393–5398, 2010.
- [73] René Thomas. Boolean formalisation of genetic control circuits. *Journal of Theoretical Biology*, 42:565–583, 1973.

Bibliography



- [74] Simon Thorpe, Arnaud Delorme, and Rufin Van Rullen. Spike-based strategies for rapid processing. *Neural networks*, 14(6):715–725, 2001.
- [75] Holger Wagner, Mark Mller, and Klaus Prank. COAST: Controllable approximative stochastic reaction algorithm. *The Journal of Chemical Physics*, 125(17):174104, 2006.
- [76] Anna Wieckowska, Nizar N Zein, Lisa M Yerian, A Rocio Lopez, Arthur J McCullough, and Ariel E Feldstein. In vivo assessment of liver cell apoptosis as a novel biomarker of disease severity in nonalcoholic fatty liver disease. *Hepatology*, 44(1):27–33, 2006.
- [77] Kazuo Yano, Yasuhiko Sasaki, Kunihito Rikino, and Koichi Seki. Top-down pass-transistor logic design. *Solid-State Circuits, IEEE Journal of*, 31(6):792–803, 1996.
- [78] Cheng Yung-Chi and William H Prusoff. Relationship between the inhibition constant (k_i) and the concentration of inhibitor which causes 50 per cent inhibition (i_{50}) of an enzymatic reaction. *Biochemical pharmacology*, 22(23):3099–3108, 1973.
- [79] David Yu Zhang and Georg Seelig. Dynamic dna nanotechnology using strand-displacement reactions. *Nature chemistry*, 3(2):103–113, 2011.
- [80] David Yu Zhang and Erik Winfree. Control of dna strand displacement kinetics using toehold exchange. *Journal of the American Chemical Society*, 131(47):17303–17314, 2009.
- [81] Guoqiang Peter Zhang. Neural networks for classification: a survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 30(4):451–462, 2000.