Building a Robust, Efficient Compiler for Gene Regulatory Networks

BE/CS/CNS 191b

Albert Ge
California Institute of Technology
Pasadena, CA
age@caltech.edu

ABSTRACT

We build a compiler using only gene regulatory networks (GRNs), using the key concept of molecular titration. We focus primarily on the problems concerning error-correction and gene complexity of the GRN, and address each issue via a greedy strategy - that is, using as much as simplicity as possible before increasing the complexity. Finally, we demonstrate as a proof-of-concept using the compiler to construct a NAND gate.

1. INTRODUCTION

GRNs (Gene Regulatory Networks) are individual gene components, linked together in a network by products of each gene affecting other genes (Figure 1). Gene components can be described as a series of chemical reactions, which describe the actions of transcribing and translating the gene. Using the concept of CRNs (chemical reaction networks) we write, for a single gene component:

$$gene \xrightarrow{k_{tx}} gene + transcript$$

$$transcript \xrightarrow{k_{tl}} transcript + protein$$

where k_{tx} describes the transcription rate, and k_{tl} is the translation rate. To allow genes to achieve a steady state, we introduce degredation rates to remove the products that are produced by the gene:

$$transcript \xrightarrow{k_{rd}} \varnothing$$

$$protein \xrightarrow{k_{pd}} \varnothing$$

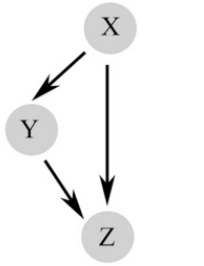
And finally, to link individual gene components to each other, we allow proteins produced by genes to bind, functionally acting as a transcription factor (TF), to other genes. The resulting gene-TF complex can then change its rate of transcribing RNA:

$$gene + TF \xrightarrow{k_{bind}} geneTF$$

$$geneTF \xrightarrow{k_{leak}} gene + TF$$

$$geneTF \xrightarrow{k'_{tx}} geneTF + transcript$$

Together, these reactions model the framework for a gene component. We can use this abstraction to build networks that can simulate computation; the input will be the starting concentrations of the system, and the output will be the final concentrations of the system. Because concentrations are a continuous quantity, we will use a similar definition



Feed-forward loop
(FFL)

Figure 1: Example of a feed-forward loop (Keifer et. al [2]). Each node represents a gene component, and each directed edge represents a gene's effects on other genes.

borrowed from digital abstraction: a LOW state, which corresponds to low concentration, and HIGH state, which corresponds to high concentration. This allows us to discretize values that we can use to model a computer working with bits of 0 or 1.

One challenge, then, is to use GRNs as a form of computation with the ability to produce error-free results. In practice, initial conditions of a system will never be exact, and there will always be small deviations of ideal values, which could potentially cause computations to fail if the system is particularly sensitive to small changes in values. Therefore, the question is, given inputs that are valid within the boundaries of our digital abstraction, to produce results that are also valid, and furthermore can be reused as inputs in downstream genes, without increasing the margin of error of the final products. This is a term we will refer as the *robustness* of GRN: that is, constructing a system that is robust to small deviations in initial conditions, and is error-correcting between individual gene components.

A second challenge is to be able to minimize the number of gene components required for a particular function. Given enough parameters, one could theoretically construct a system to perform perfectly a particular function. However, reducing the complexity of systems is favorable, as it reduces the difficulty in constructing large systems, and having fewer genes downstream means less time overall in pro-

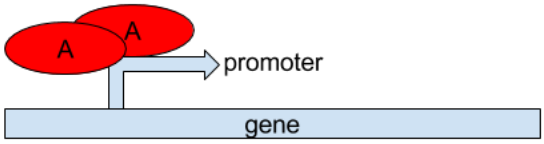


Figure 2: Dimerized gene system. Multiple TFs can bind and activate or repress the expression of a gene.

ducing a computed result. We refer to this optimization as the *gene complexity* of a system; that is, the number of genes required to perform some function. A system is said to be *gene-efficient* if, by some arbitrary standard, does not require many additional genes whose sole purpose is error-correcting. For the purposes of this paper we are mostly interested in relative efficiency; that is, given a gene system, whether it is possible to use fewer genes.

Solving these two issues would make elementary steps toward easily scalable GRN systems, which can be tasked with more complex tasks. Therefore, in this paper we explore the possibility in constructing an elementary *compiler* for GRNs: by only specifying " input" genes, their initial concentrations, and the desired "output" genes, build the corresponding gene component(s) necessary, and produce a result that corresponds to a desired function.

The rest of the paper is organized as follows. Section 2 contains details on prior work, which puts into perspective the key idea used for building robust gene systems. Section 3 presents some of the challenges encountered and the resulting solutions to overcome them. Section 4 shows a proof-of-concept of the compiler. Section 5 concludes the paper.

2. RELATED WORK

In the literature, two explored mechanisms for understanding how GRNs are robust are using dimerized factors, and using molecular titration. Systems with dimerized factors allow more than one transcription factor to bind a particular gene at the same time (Figure 2).

The greater the degree of cooperativity (i.e. number of factors bound), the greater effect of the TF. By allowing high cooperativity, the resulting dynamics follow the Hill equation, which closely resembles a sigmoid curve (Figure 3).

The sigmoid curve is one way to satisfy digital abstraction; inputs that considered "perfectly low", and those that are not perfectly low have outputs that do not differ much in value. Importantly, these functions can be applied repeatedly in sequence to make the system even more robust, as with each iteration the outputs will move closer and closer to "perfectly low". This contrasts with a simplistic linear function, after applying them in sequence, the output does not appear to improve.

There has already been a proof-of-concept shown by Winfree for using dimerized systems as a means of computation, such as building a binary counter. This work demonstrates the

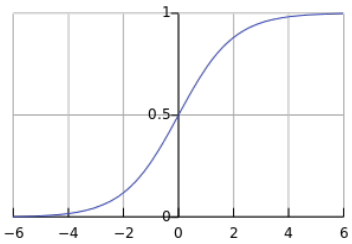


Figure 3: A sigmoid curve (courtesy of Wikipedia). Notice the nonlinearity of the function; values left of 0 are "pushed" down, whereas values right of are "lifted" up.

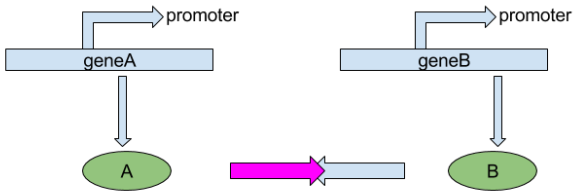


Figure 4: Titrated gene system. Each gene expresses their own construct, which then bind to each other with extremely high affinity.

computing power of GRNs, however it also mentions the inefficient gene complexity with dimerized repressors. The fundamental inefficiency lies in the ensuring the robustness of gene output; many additional genes have to be added in order to restore the output to valid ranges.

The other mechanism, molecular titration, has been championed by Buchler et al as a way to achieve "all-or-nothing" responses [1]. Molecular titration is forcing two transcription factors to bind, with extremely high affinity, to form an inactive heterocomplex (Figure 4). This allows for strong polarization of a system, as it defines a sharp boundary between two different states.

Of particular interest, then, is to examine how molecular titration matches up with dimerized repression. Therefore, the goal of this paper is use molecular titration as a primary restoring force to ensure that outputs are within valid ranges, and that deviations from perfect initial conditions still yield the same outputs.

3. CHALLENGES AND RESOLUTIONS

The following describes a series of trials attempted, in order of increasing complexity. The strategy for finding a gene-efficient representation is to first build as simplistic a framework, and then add additional complexities only when necessary. For each attempt, the successes and failures are discussed, and respectively used or addressed in subsequent attempts.

3.1 A Reporterless GRN

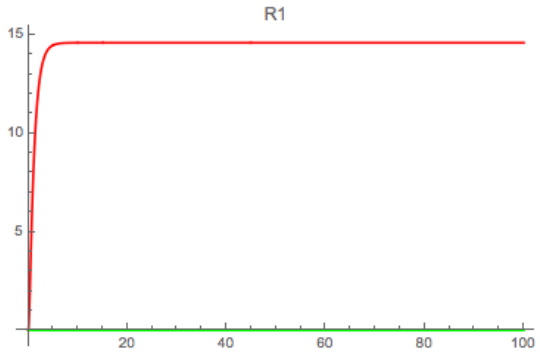


Figure 5: Example plot of the reporterless system. Here, the inputs are $k_{tx} = 177$ and $k_{tx}^c = 250$. Since there is a greater transcription rate of $C1$, only $C1$ only remains at nonzero steady state, as expected.

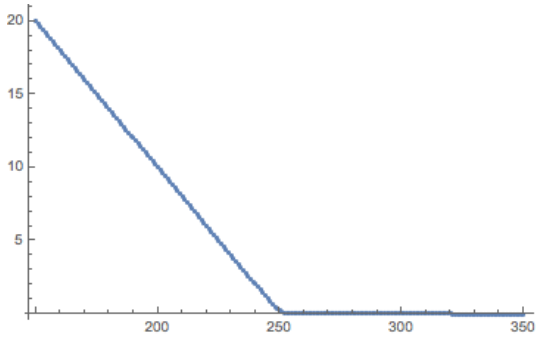


Figure 6: Plot of transcription rate k_{tx} and the steady state concentration of $C1$. k_{tx}^c was kept constant, at 250. While the plot does show a change around $k_{tx} = 250$, the plot looks mostly linear, which fails to satisfy the digital abstraction.

Consider first a GRN system with no external transcription factors, and with only two genes, $geneR1$ and $geneC1$, which produce the proteins $R1$ and $C1$, respectively. The CRN framework can be described as follows:

$$geneR1 \xrightarrow{k_{tx}} geneR1 + transcriptR1$$
$$transcriptR1 \xrightarrow{k_{tl}} transcriptR1 + R1$$
$$transcriptR1 \xrightarrow{k_{rd}} \varnothing$$
$$R1 \xrightarrow{k_{pd}} \varnothing$$

$$geneC1 \xrightarrow{k_{tx}^c} geneC1 + transcriptC1$$
$$transcriptC1 \xrightarrow{k_{tl}} transcriptC1 + C1$$
$$transcriptC1 \xrightarrow{k_{rd}} \varnothing$$
$$R1 \xrightarrow{k_{pd}} \varnothing$$

$$R1 + C1 \xrightarrow{k_c} \varnothing$$

We will refer to $geneR1$ as the *analyte* gene, and $geneC1$ as the *titrant* gene. Intuitively, these genes produce their respective proteins, and the proteins undergo an annihilation reaction with extremely fast rate k_c.

It is possible to build a function around this system, which takes as required inputs k_{tx} and k_{tx}^c, and plots the resulting simulation of $R1$ and $C1$ with these inputs. These input parameters directly govern the rates of transcription, and therefore can be varied to produce either more $R1$ or $C1$ (Figure 5).

Using the `Manipulate` function in Mathematica, a cursory glance revealed that all reactions with rates of transcription between 150 and 350 achieved steady state within a time of $t = 100$. To get a better sense of how sensitive the system was to changes in transcription rates, a number of simulations were run by varying k_{tx} and keeping k_{tx}^c constant at 250. The resulting steady states from each simulation (recording the final output value of $C1$ at $t = 100$), was then plotted as a function of k_{tx} (Figure 6).

This first attempt fails to capture same nonlinearity described by a sigmoid curve. While there is a noticeable boundary when $k_{tx} > k_{tx}^c$ and $k_{tx} < k_{tx}^c$, the plot still looks linear. This fails a digital abstraction because it is not error-correcting. Thus, we look to introduce add some complexity that allows for more nonlinear behavior.

3.2 Adding in the Reporter

The issue of linearity arises in the unboundedness of the final outputs. For $k_{tx} > k_{tx}^c$, as k_{tx} increases, the resulting concentration of $R1$ increases as well. Therefore, there must exist a downstream binding site for $R1$ to bind to, to effectively "gate" excess $R1$ from increasing without bound. Second, to achieve a sharp boundary we require that $[R1] \gg$ [binding site], or $R1$ binds to the binding site with high affinity. This allows us to take advantage of the linearity previously found; too much $R1$ production will be saturated at the binding site, and too little $R1$ production will be suppressed by $C1$, but when the production rates are similar a small amount of $R1$ is enough to saturate the binding site, providing a clear boundary of high and low values of $R1$. Therefore, the reported *fraction* of unbound gene is now the output (Figure 7).

To maintain the spirit of using GRNs we introduce a third gene which $R1$ binds to We term this gene as the "reporter" gene, which can be represented as the following CRN:

$$R1 + reporter \xrightarrow{k_{ON}} R1reporter$$
$$R1reporter \xrightarrow{k_{OFF}} R1 + reporter$$

Notice that the reaction is reversible, as this prevents any small amount of $R1$ from permanently sticking to the reporter gene.

This is a much better digital abstraction. However, the issue is that the inputs and outputs do not line up; the inputs are rate parameters, while the outputs is unbound gene. Furthermore, the valid ranges for input and output are vastly different; input is on the order of 10^2, while output is $(0, 1)$. In order to allow for easily scalable systems, it is necessary

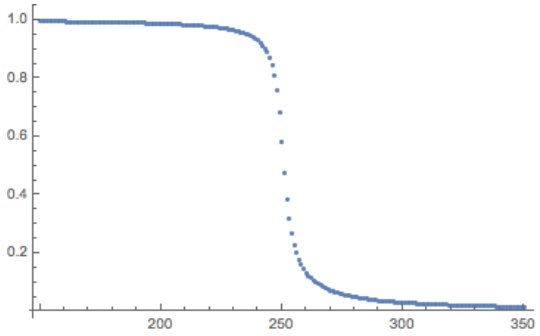


Figure 7: Plot of transcription rate k_{tx} and the fraction of unbound reporter gene. This satisfies the digital abstraction, as it discretizes low and high inputs.

to standardize both the inputs and outputs, while preserving the same nonlinear dynamics.

3.3 Building Composable Circuits I: Standardizing Input and Output

The next step is to build small components that can be easily composed; that is to say, create building blocks of GRNs that can pass along an output from one to the next. As presented earlier, one challenge is to standardize input and output (both the type, and the ranges).

To address the type mismatch, we desire to the input and output to both be a function of the concentration of protein. The output is relatively simple; just allow the reporter gene to produce a reporter protein:

$$R1reporter \xrightarrow{k_{tx}^{R}} R1reporter + reporterT$$

$$reporterT \xrightarrow{k_{tl}} reporterT + reporterP$$

$$reporterT \xrightarrow{k_{rd}} \varnothing$$

$$reporterP \xrightarrow{k_{pd}} \varnothing$$

To reduce the complexity of the system, we use the similar rate constants as before.

To address the range mismatch, we take in as input concentrations of proteins r and c, which subsequently bind to the genes for $R1$ and $C1$, respectively, and allow the bound genes to transcribe at a higher rate than that of the unbound rate:

$$r + geneR1 \xrightarrow{k_{ON}} rgeneR1$$

$$rgeneR1 \xrightarrow{k_{OFF}} r + geneR1$$

$$rgeneR1 \xrightarrow{k_{tx}} rgeneR1 + transcriptR1$$

$$c + geneC1 \xrightarrow{k_{ON}} cgeneC1$$

$$cgeneC1 \xrightarrow{k_{OFF}} c + geneC1$$

$$cgeneC1 \xrightarrow{k_{tx}} cgeneC1 + transcriptC1$$

By adjusting the internal parameters accordingly, we can achieve similar dynamics as in Figure 5, the difference being that both input and output range between 0 and 10.

3.4 Building Composable Circuits II: Building an Abstraction Layer

The second challenge in building composable circuits for the compiler is to begin abstracting away parts on a GRN. The advantage here is twofold; to be able to reuse components when building a GRN, and second to be able to build GRNs that can communicate with each other by passing along one GRN's output to the next. Here, the GRN illustrated already provides a natural division of parts; each gene (and its associated reactions) can be abstracted away as one component, so that only the input and output are visible. Additionally, each component should be allowed to take in multiple inputs, as this reduces redundancy in the number of genes used. Figure 8a presents a sample diagram of the individual components in conjunction, with only the bolded areas visible to the compiler user.

Therefore, a titration system involves three primary components: the analyte gene, the titrant gene, and the reporter gene. Each input can be specified as either an "activator" or "repressor", which will increase or decrease the normal levels of transcription, respectively. Additionally, to keep each component distinct from one another, the compiler will automatically assign new variable names to each species in the reaction based on the output of the gene. The key assumption, then, is that the user will specify each input/output species exactly once in the GRN system, so as to avoid further redundancy of species.

We have now elucidated all the key components that are utilized in our compiler. Finally, it remains to demonstrate the compiler in building a multicomponent GRN system, and show how it works.

4. PROOF-OF-CONCEPT: A NAND GATE

Because the NAND gate is a universal logic gate, using the GRN compiler for the NAND gate would pave the way into building more logic gates, and begin to shed light on constructing full-scale computational models using only genes.

We construct three genes for the NAND gate; one is the "input" gene, in which the two inputs (corresponding to the two NAND inputs) both affect that gene. The second is the "titrant" gene, which thresholds the resulting effect of the two genes. The last is the reporter gene, which produces the output of the GRN system. (Figure 8).

The intuition behind building this framework is that the two NAND inputs will both activate the input gene, resulting in an additive effect on the output produced by the input gene. The product is then titrated by the titrant output: effectively, if the concentration of product is above a certain threshold (i.e. only when the two inputs are high), only then will there be leftover product to bind to the reporter and repress output. In all other possible permutations (only one input is high, or both are low), there is not enough product to completely repress the reporter's output.

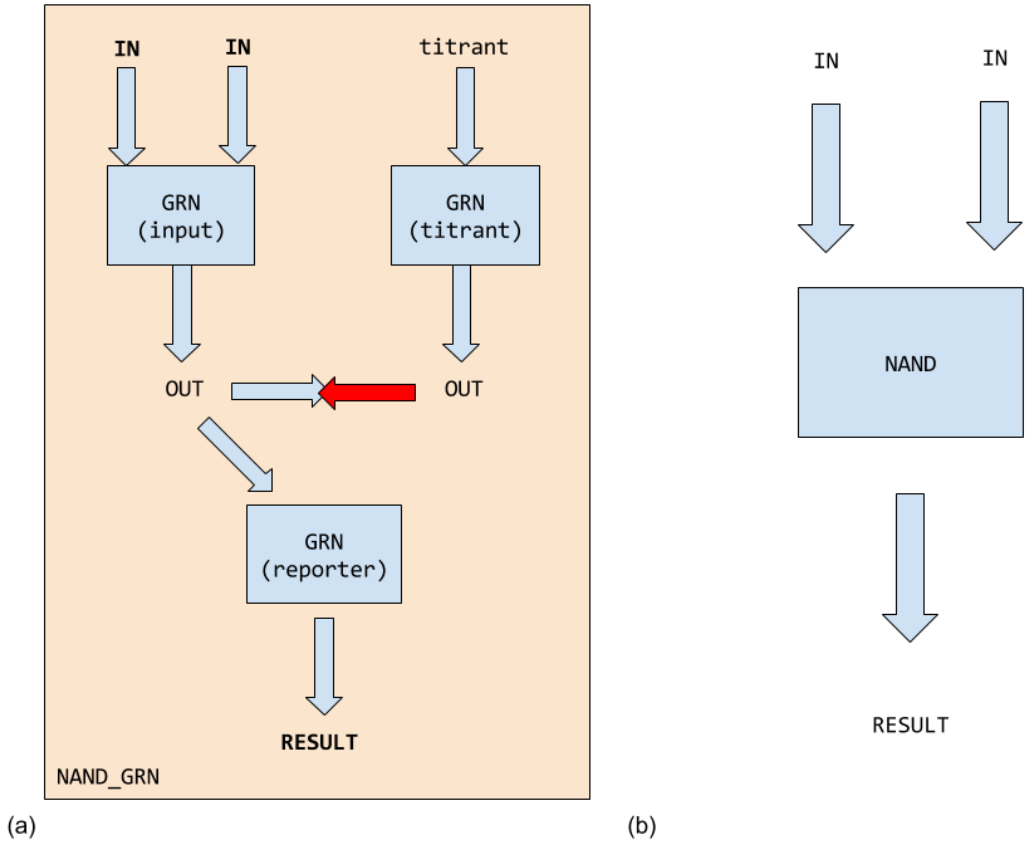


Figure 8: Schematic comparison of (a) a NAND gate modeled by a GRN framework and (b) an abstract NAND gate . There are three individual GRN components - the input, the titrant, and finally the reporter.

The time traces of the input and output species of NAND (Figure 9) shows that the GRN indeed satisfies properties of NAND, even with a fairly large error margin (20% of perfectly low or high input). Given two input protein A, B, the output of the NAND gate, Z, is only low when both inputs are high. This proof-of-concept, therefore, strongly demonstrates that the compiler, through titration, is able to compute accurately the output of a NAND gate, and suggests that it is well-suited for building more complex digital logic circuits.

5. CONCLUSION

In this paper we explain the mechanism for constructing a GRN compiler, that only requires specification of inputs and output, and is able perform computations in both a robust and efficient manner. The primary motivation for creating this compiler was to apply the concept of "molecular titration", which enforces a strict boundary between low and high concentrations of input. We provide the rationale for exploring certain possibilities to reduce the gene complexity, while also preserving error-correction. Finally, we demonstrate our compiler by modeling a NAND gate, and showing that it satisfies the digital abstraction.

Although work has been now done to elucidate the power of GRNs with titration, it still remains to see a full comparison between dimerized systems and titrant systems. In building the binary counter, Winfree has also been able to construct a NAND gate also using three genes; all three are linked together in a single pathway. However, it is noted that the binary counter is significantly more complex - on the order of 400 genes. Thus, molecular titration systems may help drive down the complexity of the binary counter.

The second is to be able to link the GRN compiler with higher abstraction layers. One such layer are the neural networks; these are clusters of networks that take in a large number of inputs, assign appropriate weights to each input, and produce a number of outputs, which can be compared to either a training set, or fed back as inputs so that the network can "learn" from each iteration. It is thought that GRNs resemble networks closely; the weights merely correspond to transcription rates of genes, and the genes can produce outputs that either are self-activating, or can activate other genes downstream. This could prove to be an interesting avenue to pursue, as neural networks have been a popular topic in computer science, and approaching them from a genetic perspective could shed more light on how to make efficient learning networks.

6. ACKNOWLEDGMENTS

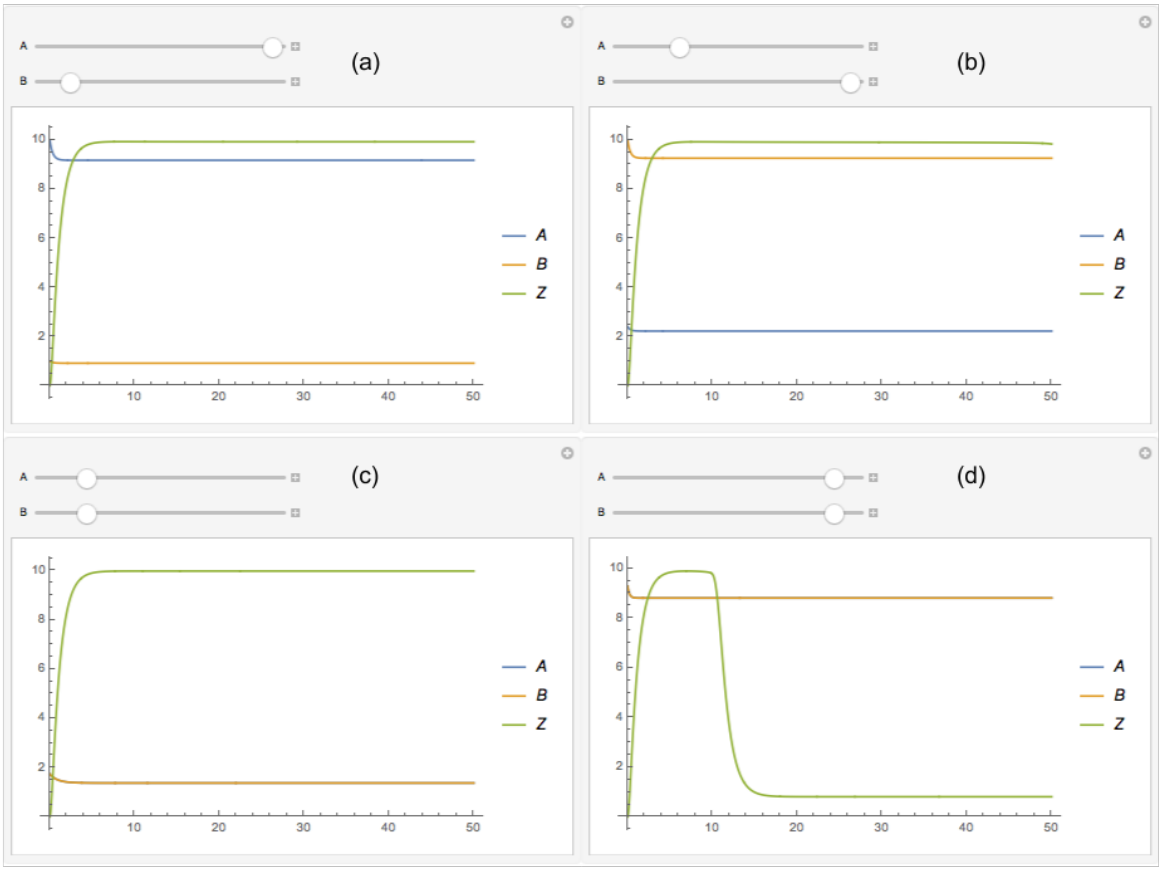


Figure 9: Time versus concentration plots of NAND gates. The two input proteins are A, B, and the output is Z. (a) and (b) describe the time dynamics with one high input and one low input. (c) describes the dynamics with both low inputs, and (d) describes the dynamics with both high inputs.

For the majority of the project, Mathematica was used as the programming language to develop and debug the project. Within the framework of Mathematica, I used David Soloveichik's CRNSimulator package.

I'd also like to thank Erik Winfree for mentoring and overall providing feedback throughout this project. Overcoming the obstacles encountered in the project would not have been possible without his guidance and suggestions, and I am very thankful for his helpful insight.

7. APPENDIX

All source code for the project can be found at `https://github.com/albertge5`. For convenience, a copy of the latest version of the compiler will be posted on the following page.

```
Clear[GRN2];
EnumTF[TF_, gene_, transcript_, ktx_]:=(
boundGene = Symbol[ToString[TF] <> ToString[gene]];
Seq[
revrxn[TF + gene, boundGene, kON, kOFF],
rxn[boundGene, boundGene + transcript, ktx]
]
)


Second[x_]:=x[[2]]

EnumConc[TF_, conc_]:=(
Cases[Tuples[{TF, conc}], {v_, v_[0] == b_} → {v, b}]
)

GRN2[IN_, OUT_, init_]:=(

active = First[#]&/@Select[IN, Second[#] == "ON"&];
repress = First[#]&/@Select[IN, Second[#] == "OFF"&];
With[{nodes = Complement[Join[active, repress], Cases[init, v_[0] == _ → v]]},
inits = Join[init, (#[0] == 0)&/@nodes]];
output = ToString[OUT];
gene = Symbol[g <> output];
transcript = Symbol[t <> output];
Seq[
Function[x, EnumTF[x, gene, transcript, kAct]]/@active,
Function[x, EnumTF[x, gene, transcript, kRep]]/@repress,
rxn[gene, gene + transcript, ktx],
rxn[transcript, transcript + OUT, ktl],
rxn[transcript, 0, krd],
rxn[OUT, 0, kpd],


Function[x, conc[First[x], Second[x]]]/@EnumConc[Join[active, repress], inits],
conc[gene, Gtot]

]
)

Clear[GRNSim2, GRNSimPlot2];
GRNSim2[GRN_, watch_, time_]:=(
rsys = Flatten[GRN];

sol =
SimulateRxnsys[
rsys/.{kON → 0.25, kOFF → 0.25, kAct → 50.0, kRep → 40.0, ktx → 0.0, ktl → 1.0,
kpd → 1.0, krd → 5.0, kc → 1000, Gtot → 1}, time];
(#[t]&/@watch)/.sol
)
GRNSimPlot2[GRN_, watch_, time_]:=(
plotter = GRNSim2[GRN, watch, time];
Plot[plotter, {t, 0, time}, PlotRange → All,
PlotLegends → watch]
)

NAND1[GRN_, inits_]:=(
inputs = Flatten[GRN];
X = inputs[[1]];
Y = inputs[[2]];
OUT = inputs[[3]];
INT = Symbol[ToString[OUT] <> "INT"];
titr = Symbol[ToString[OUT] <> T];
```

```
rsys = {
{GRN2[{{X, "ON"}, {Y, "ON"}}, INT, inits]}/.{kON → 0.25, kOFF → 0.25, kAct → 60.0},
{GRN2[{{titr, "ON"}}, titr, inits]}/.{kON → 2.0, kOFF → 1.0, kAct → 81.0},
rxn[INT, 0, kpd],
conc[titr, 5],
rxn[INT + titr, 0, kc],

{GRN2[{{INT, "OFF"}}, OUT, {INT[0] == 0, titr[0] == 0}]}/.
{kON → 2.0, kOFF → 1.0, kRep → 0.0, ktx → 50.0}
}
)


tmax = 50;
Manipulate[GRNSimPlot2[NAND1[{{A, B}, Z}, {A[0] == i, B[0] == j}], {A, B, Z}, tmax],
{{i, 1, A}, 0, 10}, {{j, 1, B}, 0, 10}]
```

8. REFERENCES

[1] N. E. Buchler and M. Louis. Molecular titration and ultrasensitivity in regulatory networks. *Journal of molecular biology*, 384(5):1106–1119, 2008.

[2] A. S. Habib, J. C. Keifer, C. O. Borel, W. D. White, and T. J. Gan. A comparison of the combination of aprepitant and dexamethasone versus the combination of ondansetron and dexamethasone for the prevention of postoperative nausea and vomiting in patients undergoing craniotomy. *Anesthesia & Analgesia*, 112(4):813–818, 2011.