

Three Problems in the Design and Specification of Biomolecular Circuits

Thesis by
Samuel Eric Clamons

In Partial Fulfillment of the Requirements for the
Degree of
Bioengineering

The logo for the California Institute of Technology (Caltech), featuring the word "Caltech" in a bold, orange, sans-serif font.

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2022
Defended September 13, 2021

© 2022

Samuel Eric Clamons
ORCID: 0000-0002-7993-2278

All rights reserved

ABSTRACT

Programming biological materials is a daunting challenge. Although part of this challenge is practical—cloning is difficult, synthesizing DNA is expensive at scale, etc.—a number of the challenges of bioengineering (and synthetic biology in particular) are problems of *design and specification*. If we could place arbitrary molecules on a surface with perfect precision, what should we place and where? If we could arbitrarily change the genetic content of a cell, even with perfect knowledge of the function and action of every component, what changes would actually enact the functions we want that cell to have? In this thesis, we explore three specific design and specification challenges at three different levels of abstraction, and demonstrate methods for overcoming them. On the level of **design language**, we use a specialized class of cellular automaton to probe what chemistry can do when restricted to a surface. On the level of **part specification**, we use several models of CRISPR/Cas9-based transcriptional regulators to understand what dynamic functions those regulators can perform and why, and provide some suggestions for how to engineer such regulators to more robustly perform those functions. On the level of **module design**, we consider an easy-to-encounter trap in when modeling a replicating DNA species in a CRN-based biocircuit simulation, for which we suggest a simple, flexible, biologically-plausible workaround.

PUBLISHED CONTENT AND CONTRIBUTIONS

Clamons, S., Qian, L., Winfree, E. (2020) “Programming and Simulating Chemical Reaction Networks on a Surface”. In *Journal of the Royal Society Interface* 17:1720190790. DOI:10.1098/rsif.2019.0790.

S.C. designed automata, designed and wrote code, performed simulations, and wrote the manuscript with critical feedback and contributions from Q.L. and E.W. Q.L. and E.W. conceived the original idea, provided critical feedback at all stages of the project, and contributed passages to the manuscript.

TABLE OF CONTENTS

Abstract	iii
Published Content and Contributions	iv
Table of Contents	v
Chapter I: Introduction	1
Chapter II: Programming and Simulating Chemical Reaction Networks on a Surface	3
2.1 Introduction	5
2.2 Review: What is a Surface CRN?	8
2.3 Dynamic Spatial Patterns	12
2.4 Continuously Active Logic Circuits	23
2.5 Manufacturing	32
2.6 Robots and Swarms	37
2.7 Conclusions	42
Chapter III: Modeling Dynamic Transcriptional CRISPRi Circuits	45
3.1 Introduction	45
3.2 A Model of CRISPRi	46
3.3 Modeling Results	50
3.4 CRISPR Activators Scale Better Than CRISPR Inhibitors	60
3.5 Engineering Requirements of CRISPRi	70
3.6 Conclusions	73
Chapter IV: How to Model Replicating DNA (and Why)	81
4.1 Introduction	81
4.2 Trivially-Replicating Plasmids Have No Well-Defined Copy Number	81
4.3 Zero-Order Replication Recovers Good Steady-State Properties	82
4.4 Biological Models of Plasmid Replication Approximately Reduce to Zero-Order Replication	83
4.5 Worked Examples	87
4.6 Discussion	91
Chapter V: Conclusions and Future Work	99
5.1 Programming and Simulating Chemical Reaction Networks on a Sur- face	99
5.2 Modeling Dynamic Transcriptional Circuits with CRISPRi	99
5.3 How to Model Replicating DNA (and why)	100

Chapter 1

INTRODUCTION

How can you use biologically-derived things—molecules, polymers, cells, materials, tissues, organisms, or populations—to solve problems? This is the principal question of bioengineering.

Bioengineering itself encompasses and intersects a number of other fields and sub-fields. Of particular interest to this thesis are molecular programming and synthetic biology. Molecular programming solves problems using custom-built biomolecules—usually, but not exclusively, nucleic acids (Hu and Niemeyer, 2019), and typically in simple in vitro contexts. Synthetic biology instead uses dynamic circuits (again usually made from nucleic acids or proteins) working in the context of a cell (Meng and Ellis, 2020; Nielsen et al., 2016), or cell-like in vitro environment (Sun et al., 2013; Garamella et al., 2016).

Biological systems are complex and flexible, and nature provides us with a seemingly boundless toolbox of pre-built tools that we can take advantage of for our own purposes. Biological machines can 3D print fantastically complex structures (Pawolski et al., 2018; Wallace, Chanut, and Voigt, 2020), spin threads stronger and lighter than steel (Foo and Kaplan, 2002), and synthesize fuels from little more than air and light (Khan and Fu, 2020). It is tempting to claim that with the right DNA sequence, a cell can do anything.

However, the diverse complexity that makes biology appealing as a substrate also makes it uniquely difficult to understand and work with. Even if there exists a DNA sequence that will solve your problem, *finding* that sequence is, typically, a beast of a problem in its own right. Our understanding of biological systems is limited both by our knowledge of those systems and our ability to predict what those systems will do.

Therefore, many of the core problems of bioengineering can be distilled into more general problems of *design*—given a function or outcome, how can you specify a biological system that implements that function or produces that outcome? The dual to the design problem is the problem of *prediction*—given a specification for a biological system, what will it do?

This thesis demonstrates three different attacks on the design problem at three different levels of abstraction: design language, part specification and use, and sub-component (module) design.

1. **Design Language:** Many frameworks for planning and analyzing engineered biological systems use design languages derived from the Chemical Reaction

Network (CRN) and its associated dynamical models, sometimes with the addition of coarse-grained spatial dynamics (i.e., reaction-diffusion networks). Although powerful, these formalisms are best suited to systems with large numbers of molecules that can diffuse and interact freely. CRN-based models are *not* well-suited to, for example, spatial patterning on a membrane, or the interactions of molecular walkers on a synthetic surface (Thubagere et al., 2017), where geometry and spatial arrangement are important. In Chapter 2, we describe a formalism called the surface chemical reaction network (surface CRN) intended to fill this gap in design language space. We use this language to specify direct (theoretical) implementations of arbitrary synchronous cellular automata, arbitrary feed-forward logic circuits, and selected examples of molecular-scale manufacturing and swarm robotics.

- 2. Part Specification and Use:** Chapter 3 concerns the capabilities and limitations of a particular system of powerful and flexible parts recently used in genetic circuit design known as CRISPR-based transcription factors. These hybrid protein/nucleic acid parts can be used much like more traditional transcription factors, but are much more flexible and, more importantly, *scalable* than natural transcription factors and their derivatives. We use a simple kinetic model of CRISPRi to reproduce several classic synthetic circuits, including a repressilator and a toggle switch, even though CRISPRi lacks the explicit cooperativity traditionally thought to be required for both of those circuits. We also show that CRISPRi has some non-desirable kinetic properties, and provide several suggestions for how to engineer CRISPRi components to make the CRISPRi system more powerful and flexible. We also consider how the scalability of CRISPR-based transcription factors is limited by the availability of a core enzyme, dCas9. In particular, we show that CRISPR-based repressors are much more strongly affected by dCas9 limitation than CRISPR-based activators, suggesting design principles for scaled-up CRISPR-based genetic networks.
- 3. Module Design:** In Chapter 4, we discuss useful ways to model a common neglected motif in biocircuit engineering—the self-replicating DNA unit (chromosome or plasmid). The mechanics of DNA replication are typically hidden by assumptions of biocircuit modeling; indeed, most models do not even explicitly acknowledge that DNA replicates at all! We argue that certain questions about DNA-based biocircuits (particularly in low-concentration, stochastic regimes) cannot be answered without explicitly considering and modeling DNA replication. We highlight a common pitfall for the unwary replication-modeler, and propose a simple replication model that avoids it. We also consider a previously-published model of ColE1 plasmid replication (Brendel and Perelson, 1993; Freudenau et al., 2015) along with a simplified version which, in some parameter regimes, reduces approximately to our proposed model. Finally, we provide examples of two different plasmid-based biocircuits using each of the three replication models.

Finally we discuss future challenges and areas of further research related to each of the three example problems in Chapter 5.

References

- Brendel, V and A S Perelson (1993). “Quantitative model of ColE1 plasmid copy number control”. In: *Journal of Molecular Biology* 4 (299), pp. 860–872. DOI: 10.1006/jmbi.1993.1092.
- Foo, Cheryl Wong Po and David Kaplan (2002). “Genetic engineering of fibrous proteins: spider dragline silk and collagen”. In: *Advanced Drug Delivery Reviews* 54 (8). DOI: [https://doi.org/10.1016/S0169-409X\(02\)00061-3](https://doi.org/10.1016/S0169-409X(02)00061-3).
- Freudenau, Inga, Petra Lutter, Ruth Baier, Martin Schleef, Hanna Bednarz, Alvaro R. Lara, and Karsten Niehaus (2015). “ColE1-plasmid production in *Escherichia coli*: Mathematical simulation and experimental validation”. In: *Frontiers in Bioengineering and Biotechnology*. DOI: 10.3389/fbioe.2015.00127.
- Garamella, Jonathan, Ryan Marshall, Mark Rustad, and Vincent Noireaux (2016). “The All E. coli TX-TL Toolbox 2.0: A Platform for Cell Free Synthetic Biology”. In: *ACS Synthetic Biology* 5 (4), pp. 344–355. DOI: <https://doi.org/10.1021/acssynbio.5b00296>.
- Hu, Yong and Christof Niemeyer (2019). “From DNA Nanotechnology to Material Systems Engineering”. In: *Advanced Materials* 31 (26). DOI: <https://doi.org/10.1002/adma.201806294>.
- Khan, Sikandar and Pengcheng Fu (2020). “Biotechnological perspectives on algae: a viable option for next generation biofuels”. In: *Current Opinion in Biotechnology* 62. DOI: <https://doi.org/10.1016/j.copbio.2019.09.020>.
- Meng, Fankang and Tom Ellis (2020). “The second decade of synthetic biology: 2010-2020”. In: *Nature Communications* 11 (5174). DOI: <https://doi.org/10.1038/s41467-020-19092-2>.
- Nielsen, Alec a K, Bryan S Der, Jonghyeon Shin, Prashant Vaidyanathan, Vanya Paralanov, Elizabeth a Strychalski, David Ross, Douglas Densmore, and Christopher a. Voigt (2016). “Genetic circuit design automation.” In: *Science (New York, N.Y.)* 352.6281, aac7341. DOI: 10.1126/science.aac7341.
- Pawolski, Damian, Christoph Heintze, Ingo Mey, Claudia Steinem, and Nils Kröger (2018). “Reconstituting the formation of hierarchically porous silica patterns using diatom biomolecules”. In: *Journal of Structural Biology* 204 (1), pp. 64–74. DOI: <https://doi.org/10.1016/j.jsb.2018.07.005>.
- Sun, Zachary, Clarmyra Hayes, Jonghyeon Shin, Filippo Caschera, Richard Murray, and Vincent Noireaux (2013). “Protocols for Implementing an *Escherichia coli* Based TX-TL Cell-Free Expression System for Synthetic Biology”. In: *Journal of Visualized Experiments* 76 (e50762). DOI: <https://doi.org/10.3791/50762>.

Thubagere, Anupama, Wei Li, Robert Johnson, Zibo Chen, Shayan Doroudi, Yae Lim Lee, Gregory Izatt, Sarah Wittman, Niranjan Srinivas, Damien Woods, Erik Winfree, and Lulu Qian (2017). “A cargo-sorting DNA robot”. In: *Science* 357, ean6558.

Wallace, Andrea, Nicolas Chanut, and Christopher Voigt (2020). “Silica Nanostructures Produced Using Diatom Peptides with Designed Post-Translational Modifications”. In: *Advanced Functional Materials* 30 (30). doi: <https://doi.org/10.1002/adfm.202000849>.

Chapter 2

PROGRAMMING AND SIMULATING CHEMICAL REACTION NETWORKS ON A SURFACE

Clamons, Samuel, Lulu Qian, and Erik Winfree (2020). “Programming and simulating chemical reaction networks on a surface”. In: *Journal of the Royal Society Interface* 17 (116). doi: 10.1098/rsif.2019.0790.

This paper grew from a rotation project with Erik Winfree and Lulu Qian, which itself was a followup to their original presentation of surface CRNs in their 2014 conference report in *DNA Computing and Molecular Programming*, “Parallel and Scalable Computation and Spatial Dynamics with DNA-Based Chemical Reaction Networks on a Surface.” That paper lays out the concept of a surface Chemical Reaction Network (surface CRN) with several of the examples from my own work (along with several others) and a proposed scheme for physically implementing arbitrary surface CRNs. It made a strong statement about the power and promise of surface CRNs, but when I began working on the project, we still had a lot of questions about the space of possible surface CRNs.

In particular, we lacked good ways to prove the behavior of a surface CRN. As Erik and Lulu had discovered with their Greenberg-Hastings automaton example (discussed in some depth in this chapter), the qualitative behavior of even a very simple surface CRN can depend on fairly complex interactions, and can be somewhat surprisingly dependent on the exact tuning of rate parameters. We knew it was possible to construct surface CRNs with exact, specifically defined behavior (for example, the digital logic circuits shown in this chapter, and the Turing machine implementations shown in the first surface CRN paper), but we wanted to be able to more effectively explore the behaviors of surface CRNs with more “emergent” properties.

The critical step in opening up a wider variety of surface CRNs for examination was the construction of a simulator that made it easy to specify a new surface CRN and watch it in action. This simulator made it practical to respond to questions of “what if...?” and “could we...?” with answers of “good question, let me go check.”

The primary goal of this paper was not to wrap up a complete story, but to encourage exploration. We have identified a number of possible paths forward in the use of surface CRNs, and we have shined a light a little bit of the way down each of those paths. We hope that by providing some enticing examples and providing a tool to make it easy to expand on our work, we will inspire other researchers to find out what’s further down some of those paths.

2.1 Introduction

Skilled chemists can do many things. They can make a dizzying variety of molecules—medicines, fuels, poisons, flavoring, detergents, paints, fertilizers, explosives. They can use combinations of molecules to build materials—fabrics, plastics, glass, concrete, metals and their alloys. They can change important properties of liquids, like salinity or acidity, or even redox potential, which lets them make a battery. They can make molecules change form or color, which gives them the ability to detect light, starch, oxygen, pH, and build air or water quality sensors. With the right salts or other dissolved compounds, they can grow beautiful crystal structures, which can be made into jewelry and pottery.

But even the most skilled chemists cannot make anything as sophisticated as life—a large collection of molecules with impressive information-processing capabilities underlying fundamental behaviors such as development, learning, and self-repair (MacLennan, 2015). The basic components of life—lipids, nucleic acids, and proteins—can all be chemically synthesized. Simple interactions between chemically synthesized molecules can carry out dynamic behaviors: oscillation or chaotic behavior (e.g., the Bray-Liebhabfsky reaction (Bray, 1921), the Belousov-Zhabotinsky reaction (Zhabotinskii, 1964; Degn, 1967b; Degn, 1967a; A. Winfree, 1984), and the Briggs-Rauscher reaction (Briggs and Rauscher, 1973)). Systems including dozens to hundreds of chemically synthesized molecules, nucleic acids in particular, can be built to perform information-processing tasks including Boolean logic computation (Stojanovic et al., 2005; Seelig et al., 2006; Qian and E. Winfree, 2011), neural network computation (Qian, E. Winfree, and Bruck, 2011; Cherry and Qian, 2018), and more (Chen, Dalchau, et al., 2013; Srinivas et al., 2017). Chemically synthesized molecules can also be programmed to self-assemble into structures with a variety of shapes (Rothmund, 2006; Douglas et al., 2009; Ke et al., 2012). However, these systems are still far less complex than what is seen in nature.

Is it possible to create much more complex chemical systems than current technologies?

Theoretically, yes. Perhaps the most well-known and well-understood formal model of chemistry is the abstract chemical reaction network (CRN), which encodes each chemical reaction in a system of chemical reactions as a mapping from a (multi)set of abstract reactant species to a (multi)set of abstract product species, along with a rate constant. CRN models differ in how these mappings are applied. Stochastic, discrete CRN models give positive integer values to molecular counts and typically treat reactions as events in a continuous-time Markov chain (Gillespie, 1977). For very large numbers of molecules, stochastic CRNs can be approximated by continuous, deterministic mass-action CRN models, in which chemicals have non-negative, real-valued concentrations that change according to a set of ordinary differential equations determined by the CRN for the system. In both types of CRN models, molecules are well mixed—every molecule can freely float around and come into contact with every other molecule in the system.

Deterministic mass-action CRNs are Turing-universal (Fages et al., 2017), suggesting that chemistry is, in principle, as powerful as any programming language and can compute anything, even when restricted to reactions that can be understood with simplistic models. Stochastic CRNs can also compute anything, although the computation is only correct most of the time—occasionally it is wrong (Cook et al., 2009; Soloveichik, Cook, et al., 2008). Alternatively, they can compute anything with guaranteed correctness if we could obtain the output of the computation only when time goes to infinity (Cummings, Doty, and Soloveichik, 2016). Conversely, some computational schemes are possible with stochastic CRNs but not in deterministic CRNs. For example, stochastic CRNs can directly represent complex probability distributions (Cappelletti et al., 2020), perform probabilistic inference (Poole et al., 2017), and use the CRN’s inherent stochasticity to help solve combinatorial search problems (E. Winfree, 2019).

Even though in theory well-mixed chemistry is sufficiently powerful for carrying out arbitrarily complex information-processing tasks, practically, it is hard to scale up the complexity beyond certain limits. For example, deterministic mass-action CRNs are Turing-universal only if concentrations can be guaranteed to have arbitrarily small relative errors, as these concentrations must encode arbitrarily precise analog values used in the computation (Fages et al., 2017). Stochastic CRNs have a different practical problem: the number of molecules required for a task grows exponentially with increasingly complex tasks (Soloveichik, Cook, et al., 2008).

One problem fundamental to both deterministic and stochastic CRNs is that the entire “program” of a CRN is encoded in the interactions between molecules, and designing a large collection of molecules to interact with each other with specificity is, in general, difficult. It is inevitable that when two or more molecules come into contact with each other in a well-mixed system, even if they are not designed to interact with each other, unwanted side reactions will occasionally occur. A larger collection of molecules necessarily has a higher probability of side reactions, simply because there are more possibilities for every molecule to interact with every other molecule. High-order side reactions can be repressed by using smaller concentrations of chemical species, but reducing the concentrations will also slow down a CRN. Thus, there is a tradeoff between the *accuracy* and *speed* of a CRN-based computation.

One possible solution to the problem of restraining undesirable interactions relies on spatial separation rather than specificity. This strategy is familiar from the world of electronic engineering: in silicon circuits, interactions between components are encoded by *spatial position*. Two AND gates, for example, do not interact unless they are physically connected by a wire, even though their design is identical and their signal carrier (electrons) is identical. Moreover, it is relatively easy to place electronic components with high accuracy, and we can leverage this power to build enormously complex devices from a remarkably small set of unique interacting component types. Biology found this solution long before engineers did. In the brain, neural circuits are largely determined by their spatial layout and connectivity;

release of the same neurotransmitter yields different results depending on where it is released. Even at the subcellular level, biology exploits spatial organization to control biochemical circuit function, with scaffold proteins being a primary example (Good, Zalatan, and Lim, 2011). Turning back to engineering efforts to program chemical systems, one could imagine taking similar advantage of spatial separation by, for example, assembling molecules into *polymers* or tethering molecules onto a *surface* so that geometry will limit which molecules can reach each other (in the best case they can only touch their immediate neighbors); this prospect has engendered a considered body of theoretical and experimental work in DNA nanotechnology (Qian, Soloveichik, and E. Winfree, 2011; Lakin and Phillips, 2011; Tai and Condon, 2019; Chandran et al., 2011; Muscat, Strauss, et al., 2013; Lakin, R. Petersen, et al., 2014; Teichmann, Kopperger, and Simmel, 2014; Ruiz et al., 2015; Chatterjee et al., 2017; Bui et al., 2018).

In the course of his explorations of reversible computing and the thermodynamics of computation, Charles Bennett exhibited a theoretical polymer CRN capable of simulating arbitrary Turing machines, with each polymer in solution acting independently and in parallel (Bennett, 1982). However, it remains unclear how to engineer the hypothetical enzymes needed in Bennett's construction. Plausible implementations of Turing-universal polymer CRNs have been proposed using DNA nanotechnology, but the constructions only work when there is a single copy of certain molecules (Qian, Soloveichik, and E. Winfree, 2011; Lakin and Phillips, 2011; Tai and Condon, 2019), which imposes significant technical challenges in practice and eliminates the possibility of parallelism. It is natural to expect that surface-based chemistry will have greater computational capabilities than polymer chemistry. In their 2014 paper (Qian and E. Winfree, 2014), Qian and Winfree proposed a simple but powerful surface CRN model, along with a plausible implementation using DNA nanotechnology, which now allows us to explore (i) what chemistry on a surface, in principle, can do, (ii) what constraints, if any, being on a surface imposes on chemistry, and most importantly, (iii) what it is *better* at doing than well-mixed CRNs and polymer CRNs.

As it happens, chemistry on a surface can do quite a lot. Like well-mixed chemistry, even simple natural surface chemistry can encode surprisingly complex behavior. In one striking example, administration of oxygen and carbon monoxide gasses to a well-defined platinum surface can induce bulk oscillations, spiral patterns, or turbulence, depending on the ratios of gasses used (Jakubith et al., 1990). Other, somewhat more complex surface-chemistry reactions can produce Sierpiński triangles (Shang et al., 2015; N. Li et al., 2017; Wang et al., 2019), labyrinthine mazes (Setvin et al., 2018), and other complex (and switchable!) patterns (C. Li et al., 2017). That such simple systems can produce rich behavior suggests that surface chemistry in general ought to be quite powerful.

Of course, different chemistries do different things—alloys of metals behave differently from thin-film liquids, which behave differently from proteins embedded in a membrane. The surface CRN model does not attempt to give an accounting of each

and every possible kind of surface-based chemistry, but rather explore the general class of *chemical reactions between spatially-constrained molecules* using a simple model. Qian & Winfree used surface CRNs to generate complex dynamic spatial patterns and showed them capable of directly emulating both Boolean logic circuits and Turing machines. We will expand on these examples and demonstrate several other things that surface CRNs can do, some of which are obvious and some not. You can follow along with our examples using our surface CRN simulator, either online or on your own computer with Python (see Box 1).

Throughout this paper, many of the existing systems we reference will use DNA nanotechnology. This is because DNA is a readily-programmed substrate, in the sense that it is straightforward to specify binding and reactions between arbitrary species, and so DNA nanotechnology provides, for the moment, many of the best examples of programmable chemistry (Zhang and Seelig, 2011; Chen, Groves, et al., 2015; Bathe and Rothmund, 2017; Scalise and Schulman, 2019). We wish to emphasize that the surface CRN framework can just as easily apply to *any* system of molecules with programmable interactions that live on a surface. We hope that in the future, this will encompass a wide range of chemistries using a wide range of substrates.

Box 1: Following along with our simulator

If you would like to follow along with the examples in this paper, you can do so online at <http://centrosome.caltech.edu/Surface\textunderscoreCRN\textunderscoreSimulator>, which presents an interface to the surface CRN simulation Python package we used for most of our design and visualization. The website has all of the examples we present, plus a few extra examples for fun. You can also define and test your own surface CRNs through the website, with the restriction that they must use a square or hex grid.

For users who prefer to run our code directly, or would like to extend it, our simulator is also available as the pip-installable Python package “surface_crns.” You can install it from our github page at <https://github.com/sclamons/surface\textunderscorecrns>. This will let you watch your surface CRNs run in “real-time,” and lets you step through simulations one reaction at a time. If you are comfortable programming with Python, you can also use our Python package to define surface CRNs with more complex custom geometry or visualization.

2.2 Review: What is a Surface CRN?

We follow the definition of a surface CRN as introduced in Qian and Winfree (Qian and E. Winfree, 2014): informally, it is a chemical reaction network in which individual molecules are localized to sites on a surface and can only interact with neighboring molecules. Alternatively, a surface CRN is an asynchronous, stochastic

cellular automaton with transition rules that resemble CRN reaction rules. The surface CRN model is *not* intended to replicate or model all of the possible behaviors of chemistry that happen on or near a surface; nor is the surface CRN model intended to accurately describe what is known in the field of surface chemistry. Rather, the surface CRN is a simple model of CRN-like chemistry in which molecular interactions are restricted geometrically, which we hope will clarify understanding and inspire construction of computational chemical systems on surfaces.

More formally, a surface CRN is a continuous-time Markov chain specified by a lattice L of connected sites $i \in L$ with a state s_i at each site i , along with a set of transition rules $r \in R$ of either the form $A \rightarrow B$ or $A + B \rightarrow C + D$, each with a reaction rate λ_r . The surface CRN model is defined for arbitrary graphs of sites, but for simplicity, we will consider only square-grid lattices in this paper (though our simulator can handle arbitrary graphs). The state at each site is always a single species, so we will often use “state” and “species” interchangeably. A species represents some smallest physical unit of the system (e.g., proteins covalently bonded to a glass slide, monomers in a plastic polymer, DNA complexes attached to a DNA origami sheet, or local lattice configurations in a crystal) such that local interactions can be represented as reactions. The connections between sites in the graph define which pairs of species are physically close enough to interact. Where applicable, an “empty” state (e.g. E) may be defined for sites which do not hold anything. Note that the state of the Markov chain under this definition encompasses the entire surface, and thus the states of all sites together.

As in traditional CRNs, we refer to the species on the left-hand side of the “ \rightarrow ” in a reaction as reactants and the species on the right-hand side of the “ \rightarrow ” as products.

Unimolecular reactions (those of the form $A \rightarrow B$) can occur at any site with a state matching the reactant. For example, the rule $ON \rightarrow OFF$ would cause any ON species to spontaneously and instantly flip to an OFF species when the reaction (stochastically) occurs. Again, a reaction rate associated with the rule determines the average frequency at which the reaction will occur.

Bimolecular reactions (those of the form $A + B \rightarrow C + D$) can occur anywhere that the two reactant species are present and next to each other according to the connectivity of the surface on which they sit. At the moment when it occurs, a bimolecular reaction simultaneously and instantaneously transforms both reactants into products. Note that surface CRNs do not have any absolute orientation, so the absolute order of the reactants does not matter; however, the orientation of the products will match the orientation of the reactants in the lattice. Thus, the rule $A + B \rightarrow X + Y$ can occur anywhere that an A is next to a B , no matter their orientations, and will always convert A into X and B into Y (but never A into Y or B into X).

Jumps in the surface CRN process correspond to reaction events. An event is defined as a tuple $(I, r, t + \Delta t)$, where I is a set of one or two sites, r is a transition rule with the same number of sites as I , t is the time of the last event (or 0 if it is the first event), and Δt is the time between the event and the previous event. At the

beginning of the process and immediately after each jump event, the possible events are all $(I, r, t + \Delta t)$ for which $\{s_i\}, i \in I$ are exactly the reactants of r , with Δt drawn from an exponential distribution with mean $1/\lambda_r$. The event with the minimum Δt then occurs at time $t + \Delta t$, at which time the reactant states at sites I are replaced by the products of r . At this point, the list of possible events is recalculated, and the process is repeated. Simulation of surface CRNs can therefore be considered a variant of the Gillespie algorithm (Gillespie, 1977) for stochastic chemical kinetics, and thus is susceptible to similar optimizations (Gibson and Bruck, 2000) (see Box 2).

It is worth acknowledging again that the surface CRN model is not general enough to accommodate all the complexities that may arise in real surface chemistry, such as reaction rates depending on distance in an irregular graph, or depending on higher-order neighborhood contexts. This simplification may make it less useful as a modeling language, but at the same time it makes it more appropriate as a molecular programming language, since the requirements for a systematic general-purpose implementation will be more easily met.

Note that the surface CRN is a discrete, stochastic model, related to stochastic reaction-diffusion systems (Hattne, Fange, and Elf, 2005; Fange et al., 2010) but with several key differences. First, whereas both surface CRNs and stochastic reaction-diffusion networks consider a similarly discrete number of molecules, in reaction-diffusion systems the positions of those molecules are within a continuous space while the surface CRN model considers discrete molecules at discrete positions on the surface. Therefore, surface CRNs naturally capture exclusion effects related to macromolecular crowding, which require the introduction of extra terms in the reaction-diffusion framework (Schöneberg and Noé, 2013). Further, reactions in reaction-diffusion formalisms typically have no local geometry (the relative positions of reactants and products are not specified, as if the solution is locally well-mixed), whereas surface CRN reactions explicitly track which reactant becomes which product and positions are preserved. Finally, while having a diffusion constant of zero would be an anomaly in a reaction-diffusion system, in a surface CRN, molecules will by default not diffuse—diffusion must be explicitly added with a reaction such as $X + E \xrightarrow{k} E + X$, where X is the the diffusing molecule, E represents an empty site, and the rate constant k determines the rate of diffusion.

Box 2: Efficient Surface CRN Simulation

Surface CRNs can be simulated using a variation of the Gillespie algorithm for simulating stochastic chemical reaction networks. Briefly, using variables from the text:

1. Initialize with a global state at time $t = 0$.
2. For each reaction r that can occur at a site or pair of sites, draw a time-to-event Δt for that reaction from an exponential distribution with mean $\frac{1}{\lambda_r}$. Schedule this reaction to occur at time $t + \Delta t$.
3. For the reaction with the smallest scheduled time, change the reactants of that reaction to its products and set $t = t + \Delta t$.
4. Repeat from step 2 until a stop condition.

The naive method redundantly recalculates time-to-events for every site and pair of neighboring sites after every reaction, with total time complexity roughly $O(N \times R)$, where N is the number of sites in the surface CRN and R is the total number of reaction events simulated. We instead leverage previously calculated next-reaction times by storing all possible reactions in a priority queue, sorted by the time at which each reaction will occur:

1. Initialize with a global state at time $t = 0$.
2. For each reaction that can occur, draw a time-to-event as described and add that reaction to a priority queue sorted by $t + \Delta t$.
3. Pop the first reaction from the queue. Change reactants to products. Set $t = t + \Delta t$.
4. Remove from the queue all reactions involving any of the same sites as the current reaction.
5. If any new reactions can occur that involve any of the same sites as the current reaction, add them to the queue as per step 2.
6. Repeat from step 3 until a stop condition.

Box 2: Efficient Surface CRN Simulation (cont.)

Popping a reaction from the queue is an $O(\log Q)$ operation (where Q is the maximum number of reactions in the queue at any given time) and checking for new reactions is constant in N , R , and Q , giving total time complexity $O(N+R \log Q)$ ignoring step 4. However, step 4 (removing outdated reactions) is linear with Q . Our simulation avoids the need for step 4 by storing the times when each reaction is issued and the times when each site was last changed. When an event is popped off the queue, if any of its reactants were updated between the reaction's issue time and the current time, the reaction is scrapped and another one popped off the queue. This optimization comes at the cost of bloating the queue with outdated reactions, which can slow dequeuing somewhat.

The move from bulk (deterministic) to low-count (stochastic) has a practical upside: real-world realizations of surface CRNs can be very, very small. Where a whole test tube's worth of chemicals might be required to accurately run a bulk-reaction chemical computer, one test tube could easily hold billions or trillions of tiny surfaces, each with its own surface CRN. In short, surface CRNs ought to be aggressively parallelizable.

Importantly, as we will see in several examples, putting chemical reactions on a surface opens up an interesting tradeoff between the difficulty of controlling specific molecular interactions and the difficulty of laying out components in a specific spatial arrangement. If one has the technology to position many molecules precisely on a surface, surface CRN designs with few species and reactions can accomplish complex tasks yet it will be relatively easy to design the few desired molecular interactions; conversely, if one has the technology to implement many desired molecular interactions precisely, there are other surface CRN designs that can leverage that ability to create precise, complex spatial patterns from a simple initial spatial arrangement.

Now, what exactly can a surface CRN do?

2.3 Dynamic Spatial Patterns

The chaos of asynchronicity

To begin with, surface CRNs can model many familiar chemical processes. Consider diffusion of gas particles in a two-dimensional (lattice) space. We define states G and S , for gas and space, respectively, along with the reaction $G + S \rightarrow S + G$. When a few G species are speckled on a field of S , the gas species will diffuse about at random on the lattice, gas-like, with a diffusion rate set by the rate of the reaction $G + S \rightarrow S + G$. With the addition of reactions that change the species identities, we have a form of stochastic reaction-diffusion system, as mentioned earlier.

Surface CRNs can also model chemistry at a more coarse-grained scale. Consider the patterns generated by the Greenberg-Hastings (GH) cellular automaton (Green-

berg and Hastings, 1978), which is a discrete version of the beautiful and dynamic Belousov-Zhabotinsky reaction (Zhabotinskii, 1964; Field, Koros, and Noyes, 1972; A. Winfree, 1984). The Greenberg-Hastings cellular automaton is a discrete, synchronous, deterministic automaton where each cell has one of three states: Q (quiescent), A (active), or R (refractory). At each step of the automaton: any quiescent cell next to an active cell becomes active; any active cell becomes refractory; and any refractory cell becomes quiescent. Depending on the automaton's initial configuration, it can produce single waves, infinite spirals, or complex spiral patterns (Figure 2.1c-e, middle-left column).

The GH automaton can be modeled by a three-reaction surface CRN (Figure 2.1a). This CRN has the property that if all possible reactions were examined simultaneously, and all species changes were applied simultaneously, then the behavior of the synchronous GH cellular automaton would be reproduced exactly. However, as surface CRNs have asynchronous local updates, the GH surface CRN is stochastic and its qualitative behavior depends on each reaction's exact rate constant. With the rates shown in Figure 2.1a, the surface CRN approximates some features of the behavior of the GH automaton, but the waves and spirals produced in the surface CRN version of the GH system are thick and uneven, with ragged edges.

The GH surface CRN lacks the strict synchronicity of the GH automaton, which is what allows the original GH automaton to produce fine-grained, structured patterns like the spiral wave (see Figure 2.1d). Structures in the CRN are brittle—a clean wave front will quickly break apart as some parts of the wave spread more rapidly than others. This causes the wave front to break up, peter out, or double back on itself, so that a spiral that lasts forever in the synchronous GH automaton will be a temporary storm in the surface CRN. In this case, at least, asynchronicity in reactions washes out fine structure in the surface CRN.

The GH cellular automaton can be thought of not just as a specific model of the Belousov-Zhabotinsky reaction, but rather as a generic model of *excitable media* that support propagation of temporary activity through an otherwise quiescent space. Systems as diverse as metal surface oxidation, protein signaling on oocyte membranes, neural action potentials, heart muscle contractions, slime mold aggregation, lichen growth, forest fires, infectious disease outbreaks, and star formation, all can exhibit spiral waves and have been studied as excitable media (A. Winfree, 2001; Grassberger and Kantz, 1991; Ballegooijen and Boerlijst, 2004; Tan et al., 2020). Interestingly, well-known stochastic spatial models of forest fire and infectious disease propagation correspond closely to the GH surface CRN, with the Q , A , and R states corresponding to green, burning, and burnt trees, or to susceptible, infected, and resistant hosts, respectively (Grassberger and Kantz, 1991; Ballegooijen and Boerlijst, 2004).

Going back to the example of the Greenberg-Hastings model, we saw that an inherently asynchronous surface CRN implementation of the same local logic (and with well-tuned rate constants) can generate some similar kinds of behaviors as the original synchronous cellular automata (e.g., wave propagation), while other

important features (e.g., wave fronts remaining unbroken) fail in the asynchronous surface CRN implementation. These differences can be understood in terms of conserved quantities that the synchronous updates preserve (Greenberg, Greene, and Hastings, 1980), but which are not preserved by the asynchronous updates (see Box 3). More generally, it has long been known in the cellular automaton literature that asynchronous updates often destroy behaviors that were of great interest in the synchronous case (Ingerson and Buvel, 1984; Schönfish and Roos, 1999). To produce, say, the dazzlingly complex and specific phenomena of Conway's Game of Life (Poundstone, 1985) (an example to which we will return shortly), some kind of synchronization between reactions seems necessary. Can a surface CRN, which is intrinsically asynchronous, emulate the function of a truly synchronous cellular automaton?

Box 3: Conserved Quantities of Surface CRNs

The GH cellular automaton has a conserved property called the “winding number” that is important for several proofs of its behavior. Pick any directed, closed path through lattice sites in a GH automaton. That path will cross through zero or more “cycles” of $Q \rightarrow A \rightarrow R$ (in either direction). The net number of such cycles the path crosses (forward minus reverse) is its winding number. A simple way to compute the winding number is to traverse the path, tallying +1 for each forward step, -1 for each backward step, and then dividing the total by 3. As an example, we write down the states along a possible path, and below them, the tallies for going from the given cell to the next cell:

AQQQAARRARRQAR
-00+0+0--0++++-

The tallies sum to 3 so the winding number is 1.

Interestingly, the winding number of any particular spatial path in a GH automaton with a fixed initial condition will remain the same as the automaton progresses, regardless of the initial configuration of the automaton or the topology of the lattice (eg., square, hexagonal, triangular, irregular, etc.). This conserved quantity can be used to prove, for example, conditions under which a particular GH automaton will continue indefinitely or peter out to a uniform quiescent state.

- Does the surface CRN version of the GH automaton preserve winding number? Why or why not?
- Can you identify any other conserved quantities in the surface CRN GH automaton, or modify the reactions such that analogous quantities are conserved?

You may wish to revisit these exercises for the other surface CRN implementations of the GH automaton given in later sections.

One-to-one “spinning arrow” construction of locally-synchronous automata

Yes, it can, at least in the most important senses. Again, the cellular automata literature provides some guidance: many techniques have been invented whereby an asynchronous cellular automaton can, by incorporating extra states and extra rules, enforce that the information flow of the synchronous system remains intact within the asynchronous system (Nakamura, 1981; Gács, 2001; Lee et al., 2005). However, although the surface CRN model can be seen as a subclass of asynchronous cellular automata, the fact that surface CRN updates involve just two sites at a time, and are orientation-invariant, poses significant constraints that make direct application of prior techniques impossible—but as we will show, mechanisms with a similar spirit do work.

One way for a surface CRN to emulate the function of a synchronous automaton is for each site to a) accumulate information about all of its neighbors and b) update itself based on that information in a way that c) guarantees that it will not update itself before all of its neighbors have learned its state (so that its neighbors will not get stuck, unable to update, or update based on the wrong information about the cell). In general, this requires careful setup of both the transition rules and initial conditions of the surface CRN, particularly to avoid deadlocking. Here we will describe such a scheme for emulating any cellular automaton with a square-grid geometry and Von Neumann neighborhood (i.e., one where each cell only considers the states of the four neighboring cells with which it shares an edge, but not the ones that border on only a corner). The basic principle is that the identity of the state at each site, in addition to specifying the current state of the cell that it is emulating, also has a directional arrow pointing to the next site it needs information from. When two sites' arrows point toward each other, both sites spin their arrows in opposite directions (e.g., both clockwise) and update an internal label that tracks what information that site has gathered from its neighbors. Once a site's arrow has spun once all the way around, a set of unimolecular rules changes that site's state according to the information it has gathered from its neighbors, resetting the arrow and preparing it for the next update. Note that care must be taken in the initial setup of the CRN so that every cell's arrows *do* spin completely without deadlocking (see Box 4).

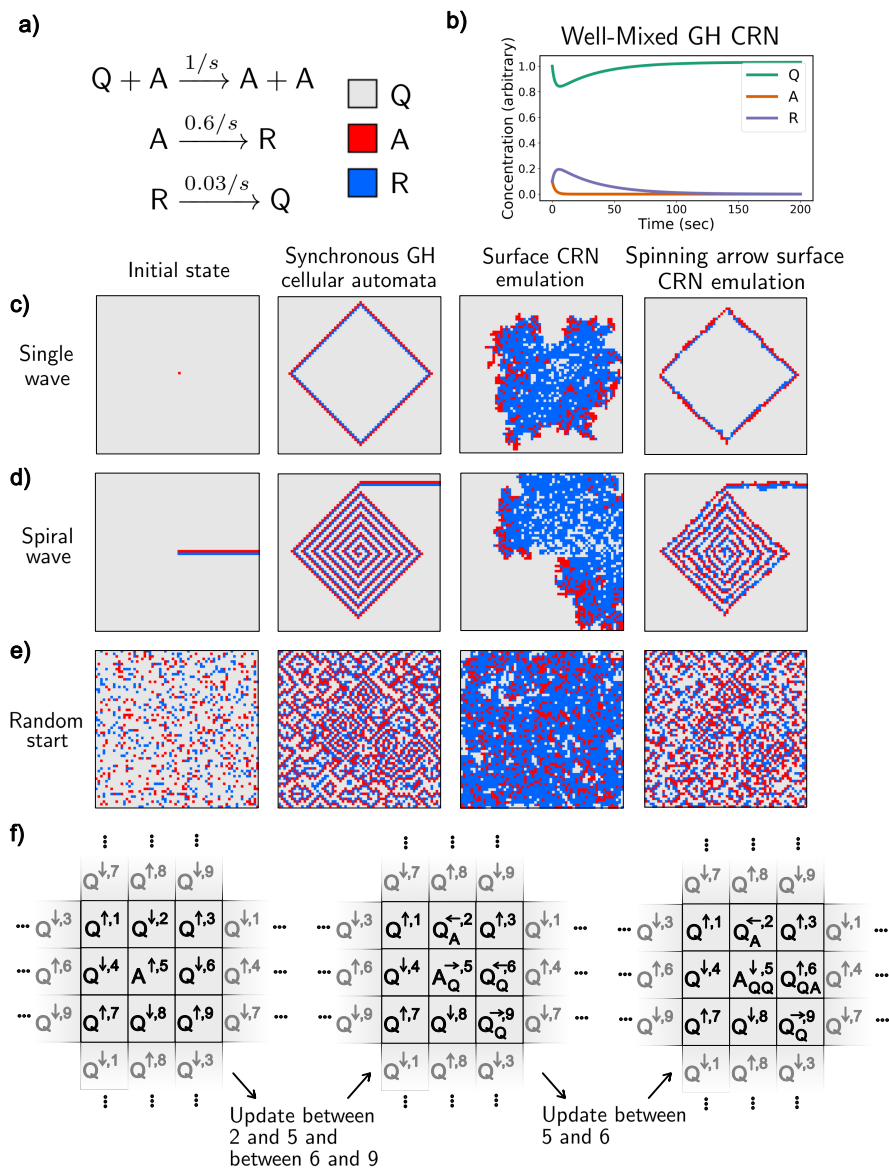


Figure 2.1: Comparison of a Greenberg-Hastings three-state excitable media cellular automaton with two surface CRN implementations. **a)** The transition rules for the surface CRN, with color legend for snapshots. **b)** Behavior of the Greenberg-Hastings system implemented as a well-mixed CRN. In bulk, this CRN displays no oscillations or other notable features. **c-e)** Initial state (left column) and snapshots for the GH automaton (middle-left column) or the direct surface CRN implementation (middle-right column) or the spinning-arrow surface CRN implementation (right column). Examples are shown for a single pulse wave (**c**), a spiral wave (**d**), and a random initialization (**e**). **f)** The spinning-arrow construction for a (locally) synchronous Greenberg-Hastings automaton using a surface CRN. Each box is a single site in the surface CRN, which emulates the function of a single cell in a synchronous automaton. At the left, arrows are shown in initial positions, arranged to avoid deadlock. The middle shows the surface CRN after two updates: one between sites 2 and 5, and another between sites 6 and 9. Sites 5 and 6 are now ready to react together. At right, we see the result of the 5-6 reaction.

Critically, each cell cannot update its state until it has received state information from all of its neighbors, and it cannot receive information from its neighbors without simultaneously sending its own state information. The result is that a cell can never be more than one “clock tick” ahead of any of its neighbors, and will only send its state information when it is at the *same* “clock tick” as its neighbor. It is *locally* synchronous. Note that there is no guarantee of *exact* synchronicity between distant cells in a locally-synchronous automaton (although cells distance d from each other can be at most d “clock ticks” apart). Nevertheless, because each cell only communicates with other cells at the same “clock tick,” the state history of each cell is guaranteed to match that of an equivalent cell in a globally-synchronous cellular automaton. In other words, the system will behave as though it were a synchronous automaton, but with some variation in the speed at which each cell updates.

One immediate problem with the spinning arrow scheme is that species in a surface CRN do not perceive local orientation. Sites can be next to each other or not next to each other, but reactions cannot be restricted based on reactants being “to the left of” or “to the right of” each other, so two arrows “pointing toward each other” could just as easily be “pointing away from each other” and the same reactions will apply. To introduce a notion of directionality to the scheme, we initialize the surface in a pattern with location labels at each site, dividing the grid into a repeating 3×3 grid (see Figure 2.1f). Adding site labels allows reactions to distinguish arrows pointing together from arrows pointing apart; for example, \leftarrow at a 2 site and \rightarrow at a 1 site point toward each other, while \leftarrow at site 2 and \rightarrow at a site 3 point away from each other.

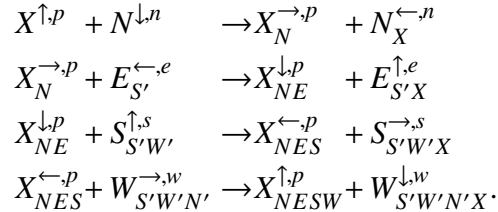
So, each site in the spinning-arrow emulation has a state that encodes:

- The current state of the emulated cell in the synchronous automaton.
- Local positional information within a 3×3 repeating grid.
- The direction of an “arrow” pointing toward the next site to update from.
- A memory recording information that the site has received from neighbors during this update cycle.

More formally, consider any synchronous cellular automaton with Von Neumann neighborhood, k states $\{s_1, s_2, \dots, s_k\} = \mathbb{S}$, and an update rule $f(\text{self}, N, E, W, S)$ where $\text{self} \in \mathbb{S}$ is the state of a cell and $N, E, W, S \in \mathbb{S}$ are the states of the cell’s northern, eastern, western, and southern neighbors, respectively. Each site in the emulating surface CRN has a state of the form $X_{info}^{d,p}$, where $X \in \mathbb{S}$ is the state of the emulated automaton at that site, $d \in \{\leftarrow, \uparrow, \downarrow, \rightarrow\}$ is the direction of the site’s arrow, $p \in \{1, \dots, 9\}$ is the local position of the site, and $info \in \mathbb{S}^*$ is a list of collected information about the states of the northern, eastern, western, and/or

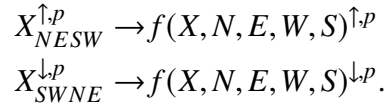
southern neighbors of the site that have exchanged information with the site, in an order based on the order of exchange.

State exchange is mediated by bimolecular reactions of the following form, where $X \in \mathbb{S}$ is an emulated state at some position $p \in \{1, \dots, 9\}$, N, E, S, W are the emulated states to the north, east, south, and west of p at positions $n, e, s, w \in \{1, \dots, 9\}$, respectively:



While these reactions at first appear to treat only a subset of possible local configurations, these are the only cases that arise if the surface CRN starts with arrows and position labels organized as shown at the left of Figure 2.1f.

Finally, states are updated using unimolecular rules of the form:



The above reactions of all forms should be defined for all appropriate pairings of $p \in \{1, \dots, 9\}$ and $n, e, w, s \in \{1, \dots, 9\}$ and $X, N, E, W, S, N', E', W', S' \in \mathbb{S}$.

By construction, only one reaction is ever possible at a time at each site, and this reaction choice is unchanged by updates elsewhere. Consequently, the rate of each chemical reaction can only affect the timing of the surface CRN's evolution, not its ultimate results. Thus, for this and other similar examples, we will ignore reaction rates (or, equivalently, assume that each reaction has rate 1).

As a specific case, consider a spiral wave in a (truly) synchronous Greenberg-Hastings cellular automaton (Figure 2.1c, middle-left) and in an equivalent locally-synchronous surface CRN implemented as above (Figure 2.1c, right). Under the right initial conditions, the spiral patterns characteristic of the fully synchronous automaton are clearly visible, but they are “fuzzy” because the molecules implementing the synchronous automaton are updated inherently randomly. No site will ever get “too far” ahead of its neighbors, or cause them to update in an order different from that of the synchronous version, but there is no guarantee of global synchronicity.

Although this example shows that (local) synchronicity is possible in a surface CRN, it is not a very satisfying construction. All of the logic handling memory storage, state updating, and arrow-spinning has to be hard-coded by “brute force”

into the species and reactions defining the circuit CRN. Generally speaking, a locally-synchronous spinning-arrow surface CRN representing a cellular automaton with k possible states requires a number of species that scales as k^5 and a number of reactions that scales as k^8 . For scale, the synchronous GH automata uses 70,794 reaction rules, not including the boundary reactions required for finite grids. With further optimization that uses species subscripts to store only the information needed by f rather than the full neighborhood state, GH can be implemented with a little over 3,000 reactions. Still, this is not a user-friendly set of reaction rules.

Box 4: Spinning Arrow Deadlocks

The spinning-arrow emulation scheme shown in this section requires careful arrangement of initial conditions—with the wrong starting arrangement, arrows can easily become deadlocked, unable to update.

- Can you prove that the initial configuration shown in Figure 2.1f will not deadlock?
- How many initial configurations are there, in terms of initial arrow direction, that avoid deadlock?
- If the surface is not infinite, then extra species and reactions are needed to mark the boundary and perform updates there that will not induce deadlock. Can you design them?
- Can you adapt the spinning-arrow emulation for surface CRNs with a hex grid graph (instead of a square grid)?

Several-to-one “broadcast-swap-sum” construction of locally synchronous automata

Another way to emulate a synchronous automaton with a locally-synchronous surface CRN is to emulate each synchronous automaton cell with an array of multiple sites in the surface CRN. A several-to-one scheme also allows simulating cellular automata with Moore neighborhoods (i.e., cellular automata whose cells communicate along corners as well as edges) without requiring direct communication across corners. We will also see that splitting a single cell into multiple sites allows us to separate the logic of site-to-site communication from the logic of site updating, which allows us to emulate the same automaton using dramatically fewer transition rules.

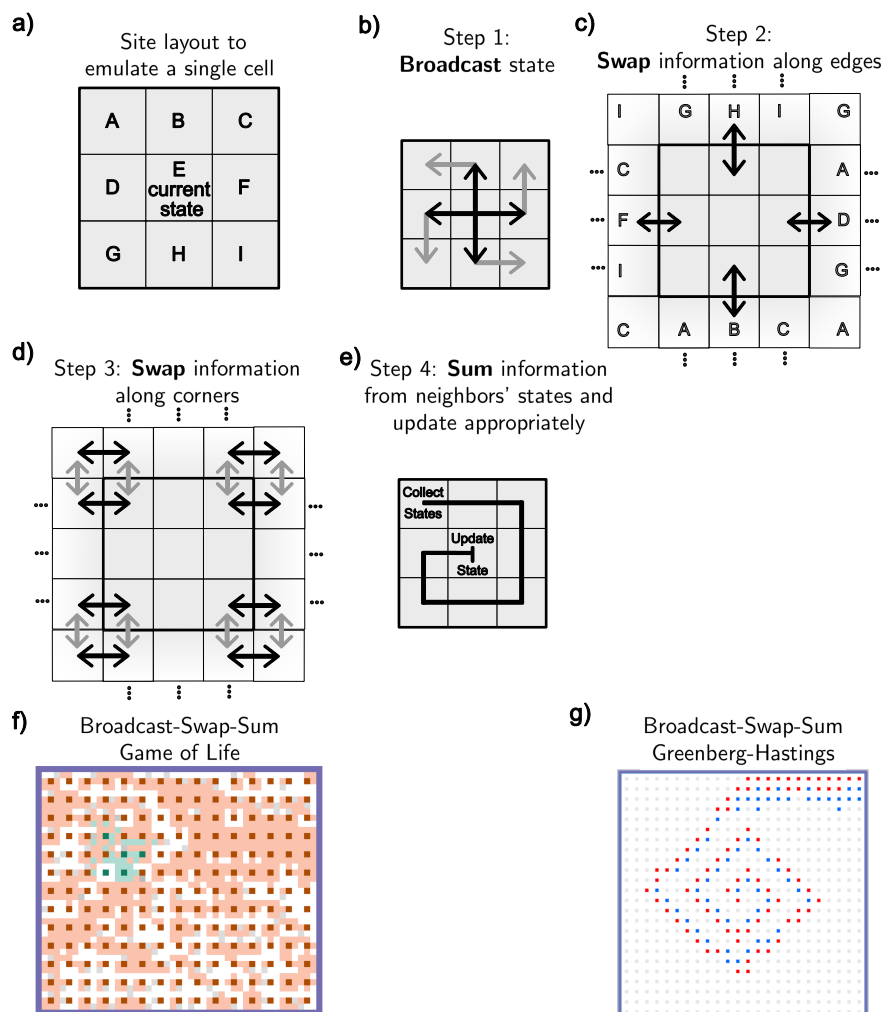


Figure 2.2: A “broadcast-swap-sum” construction of locally synchronous automata. **a)** Nine molecules emulating a single cell in a synchronous cellular automaton. The current state is held in position E, while the other molecules handle communication with neighboring cell blocks. **b-e)** A general scheme for updating such an automaton. Arrows represent information flow at different stages of the update. **f)** Simulation of a glider from Conway’s Game of Life, implemented with a 3×3 broadcast-swap-sum surface CRN emulation. Dark red sites represent “dead” cells; dark green sites are “alive” cells; light red and green sites are “wire” sites carrying information from “dead” and “alive” cells, respectively; gray sites are “wire” sites in the process of counting neighboring live cells; white sites are “wire” cells waiting for information from their corresponding sites; and purple sites represent an edge condition. **g)** Simulation of a locally-synchronous spiral wave using a broadcast-swap-sum emulation of a Greenberg-Hastings automaton, with simplified colors to make the spiral easier to visualize.

For example, consider a surface CRN implementation of Conway’s Game of Life, an automaton that uses information with a more complex update rule than the Greenberg-Hastings automaton (Poundstone, 1985). In the Game of Life, each cell represents the state of a population in some small area. The population can take one of two values—“alive” (1) or “dead” (0). The Game of Life automaton executes

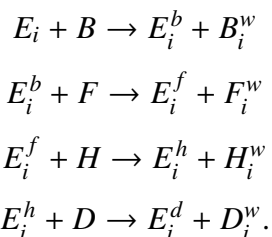
synchronously, with the new state at each position determined by its previous state and number of “alive” cells among its neighbors (including cells touching by a corner) in the following way:

- Any live cell with 0 or 1 live neighbors dies (“underpopulation”).
- Any live cell with 2 or 3 live neighbors survives and remains alive.
- Any live cell with 4 or more live neighbors dies (“overpopulation”).
- Any dead cell with exactly three neighbors becomes alive (“colonization”).
- Any other dead cell remains dead.

In a broadcast-swap-sum emulation, each cell in the Game of Life is represented by a 3×3 grid of sites on the surface CRN, labeled A through I from top to bottom, left to right (Figure 2.2a). The center molecule (position E) stores the current state of the Game of Life cell (0 or 1). The 8-molecule ring around E handles signal communication and processing.

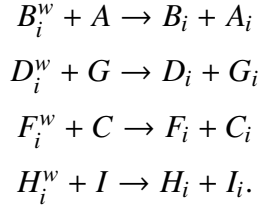
We will write states in this scheme in the form X_i^S . $X \in \{A, B, \dots, I\}$ indicates the position within each 3×3 block that the site occupies. $S \in \{\emptyset, b, f, h, d, w, n, s\}$ is a flag indicating what step of update computation the site has reached. $i \in \{0, 1, 2, 3, 4\}$ is a number representing either the state of the emulated cell or the current count in a running sum of the cell’s neighbors’ states.

Update of a node begins with site E “transmitting” its state to positions B, D, F, and H (Figure 2.2b) using the following transition rules:

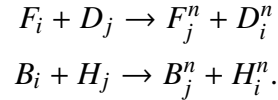


Here, the flags $b, f, h,$ and d in the species’ names indicate the last position that was updated; w indicates a state that has received information from site E but that has not yet transmitted that information to the corner sites; and $i \in \{0, 1\}$ is the current state of the emulated cell.

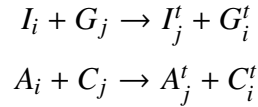
Next, similar rules transmit the emulated cell’s state from positions B, D, F, and H to positions A, C, G, and I, respectively, and set positions B, D, F, and H to states $B_i, D_i, F_i,$ and H_i , indicating that those sites are ready to transmit their current state information to neighboring 3×3 blocks.



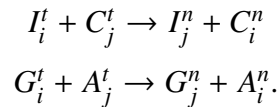
Now emulated cells can “talk” horizontally by swapping the states in positions F and D, and vertically by swapping the states in positions B and H (Figure 2.2c), using the reactions



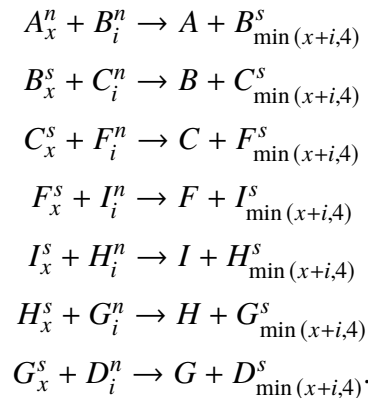
where $i, j \in \{0, 1\}$ are the states of each emulated cell and the n flag indicates that the site has received information from the neighboring emulated cell. Concurrently, corners communicate in a similar fashion. However, since corners cannot directly communicate with one another, corner sites swap states in a two-step fashion (Figure 2.2d), first swapping horizontally



and then vertically



After swapping along sides and corners, each of the eight border sites will have an n flag and will contain the state of one of the emulated cell’s eight neighbors. At this point, the total number of active neighbors is summed in a circular fashion (Figure 2.2e) using reactions of the form



Here, x is a running sum of the number of neighboring emulated cells with state 1, which is capped at 4 for compression since the behavior of a cell of the Game of Life does not change past 4 neighboring “alive” cells. The s flag indicates that the site currently holds the running sum for the emulated cell. This set of reactions also resets the ring and prepares it for the next round.

Finally, the sum in position D is combined with the current state, setting the new state and resetting D, by a set of rules of the form $D_x^s + E_i^d \rightarrow D + E_{G(x,i)}$, where $G(x,i)$ is the update rule for the Game of Life. That is, 1 if $x = 3$ or if $i = 1$ and $x = 2$, otherwise 0. An example of a glider from the Game of Life is shown in Figure 2.2f (dark green center squares).

A similar construction can be used to make a locally synchronous Greenberg-Hastings automaton with a broadcast-swap-sum emulation (Figure 2.2g). This same strategy can be used to emulate any Moore-neighborhood totalistic synchronous cellular automaton (i.e., those automata for which the update rule depends only on the current state of a cell and the *total* number of neighbors with some state), or, with a more complex neighbor-summarization procedure (which would require more reactions and chemical species, but not more physical space), any Moore-neighborhood non-totalistic cellular automaton.

Altogether, including reactions to handle boundary conditions, the broadcast-swap-sum Game of Life automaton requires 132 rules, and nine sites for each Game of Life cell, and the broadcast-swap-sum Greenberg-Hastings automaton requires only 102 reactions and 104 states, a considerable improvement over the spinning-arrow emulation technique in terms of number of transition rules required to implement the behavior. This efficiency of transition rules (which we might expect to be of prime importance in any laboratory implementation of a surface CRN) comes at two costs: larger surface CRN area per area of emulated cellular automaton, and more reaction steps per cellular automaton update.

2.4 Continuously Active Logic Circuits

A properly constructed Game of Life of sufficient size can either simulate a Turing machine (Rendell, 2002) or a Boolean circuit using streams of gliders to represent data lines (Rennard, 2002). Since the surface CRN formalism can emulate the Game of Life, it is therefore Turing-complete, which means that, in principle, surface chemistry ought to be capable of performing any computation accessible to a digital computer.

If desired, one could program a Turing machine directly as a surface CRN, as shown by Qian and Winfree (Qian and E. Winfree, 2014). In the same paper, those authors also provide a scheme for implementing arbitrary feed-forward Boolean logic circuits. These circuits function similarly to digital electronic logic circuits, with a few notable differences, as you will see below. A strength of Boolean logic circuits is that they can compute in parallel, allowing them to make decisions faster than equivalent Turing machines. We will show some interesting examples of circuits made using the strategy described in (Qian and E. Winfree, 2014), updated

slightly to accommodate feedback circuits in cases where the original strategy could produce permanent “traffic jams” at wire crosses. (Also c.f. (Toffoli and Margolus, 1987) and (Lee et al., 2005) for Boolean circuits implemented as cellular automata.)

To review, our Boolean logic circuits are based around signal states 0 and 1, which diffuse along linear paths of wire sites B (“Blank wire”) that are embedded in a field of inert states I (Figure 2.3a). Information thus flows as discrete packets that perform a random walk along wires using two diffusion reactions $0 + B \rightarrow B + 0$ and $1 + B \rightarrow B + 1$. For the sake of brevity, we will write this pair of reactions as $0/1 + B \rightarrow B + 0/1$. We will continue to use this notational convention—any rule written with a “/” represents two logically related reactions, compressed for space.

To compute, we use logic gates constructed from several adjacent sites representing input and output positions. The simplest gate we construct is the NOT gate, shown in Figure 2.3b. The NOT gate is constructed from several states of the form SNL , where $S \in \{B, 0, 1\}$ represents the data currently held at this position (either blank, 0, or 1), N designates the state as belonging to a NOT gate, and $L \in \{x, y\}$ distinguishes the input and output locations. This gate uses three sets of reactions: one to load incoming data packets (either “0” or “1”) from a wire onto the front half of the gate; one to invert those packets and move them to the back half of the gate; and one to unload the inverted packet from the back half of the gate to another wire. For example, when the front half of a NOT gate is in a waiting state BNx and is next to an incoming 0 packet, the rule $0 + BNx \rightarrow B + 0Nx$ loads the zero signal onto the front of the NOT gate and removes it from the wire. Similarly, a 1 can be loaded using the reaction $1 + BNx \rightarrow B + 1Nx$. Note that unlike wires, which allow bidirectional information flow, NOT gates only allow signals to pass in one direction, causing computation to ratchet forward.

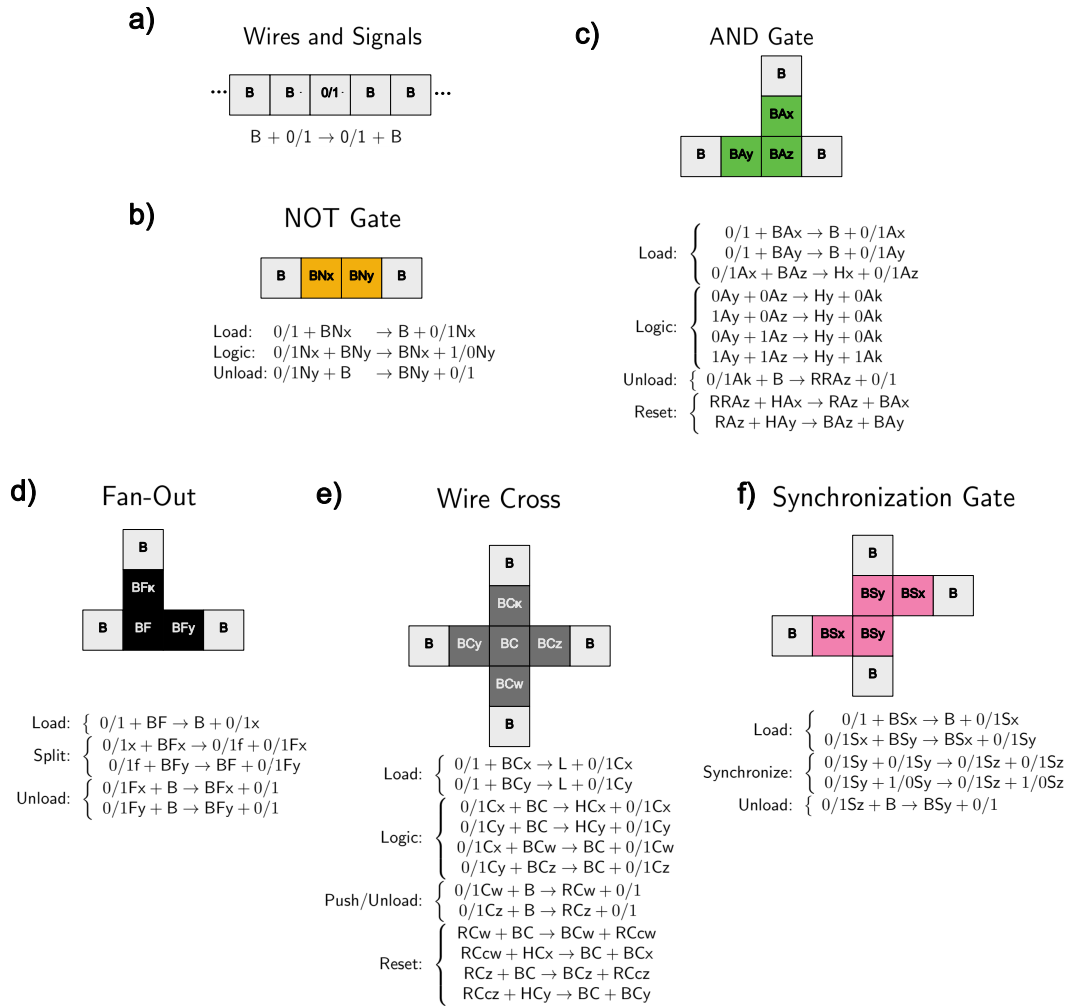


Figure 2.3: Continuously active logic circuits. B represents a blank wire, and 0 and 1 are states representing data packets. Gate locations use states with a three-symbol name, where the first symbol denotes the data at that position (either blank, 0, or 1), the second symbol distinguishes different gates, and the third symbol distinguishes different parts of the gate. **a)** Wires, signals, and signal diffusion, **b)** NOT gate, **c)** AND gate, **d)** wire fan-out, **e)** crossing wires, and **f)** synchronization gate implemented with surface CRNs.

A two-input logic gate can be built with an architecture similar to that of the NOT gate. A 14-reaction AND gate is shown in Figure 2.3c. Four rules load incoming signals onto the A_x and A_y positions. Two more rules move the A_x signal onto the A_z position, putting the A_x position into a holding state (HA_x). By combining the signals from the A_x and A_y lines, four more rules calculate the correct output signal (now on the A_z position). The output is then unloaded onto a wire, leaving the A_z position in a reset state RA_z , which triggers two reactions that reset the gate and prepare it for new incoming signals. The output signal can now diffuse away, but cannot be re-loaded into the gate. Note that the AND logic occurs entirely in the four rules that combine the A_x and A_y signals—the gate can be changed to implement other logic simply by changing these four reactions. We will use OR and XOR gates in later examples built in this way.

So far, a single wire in this implementation cannot feed a signal to more than one gate (in parallel), as each data packet (0 or 1) is consumed when it is detected by a gate. To make circuits where one line can feed to multiple gate inputs, we will need an explicit fan-out mechanism. A gate implementing a 2-output fan-out is shown in Figure 2.3d. It uses reactions similar to those of the AND gate but with opposite flow—two rules read a signal into the center of the gate, four rules “split” the signal into the two output positions, and four more rules unload each output position’s signal onto its respective wire. A 3-output fan-out can be built in a similar fashion using 14 reactions.

Finally, in order to build circuits of any reasonable complexity on a two-dimensional grid, we will need a mechanism that allows wires to cross without mixing up their signals. Figure 2.3e shows an example wire cross “gate.” For each axis (horizontal or vertical), the wire cross requires two rules to load a signal on the front end, two rules to push the signal to the center (labeled so the gate “knows” whether the signal is coming from C_x or C_y), two rules to push the signal to an output gate, two gates to unload the signal onto a wire, and two rules to send a reset signal to the front of the gate to prepare it for the next input.

Note that here (unlike in the logic gates shown earlier), holding and resetting the input lines is important for guaranteed wire cross function, at least for use in recurrent circuits. A circuit can be laid out such that both of the wires of the wire cross feed the same two-input gate. In such a case, a wire cross (such as the one presented in (Qian and E. Winfree, 2014)) without gate-locking or similar precautions can become permanently jammed if enough inputs arrive from one line to fill up the wire cross, blocking the inputs required to allow the backed-up line to proceed.

Finally, a gate that is not strictly necessary, but that will be helpful for a later construction, is what we term a “synchronization gate.” This gate is essentially two linked repeater gates (NOT gates with the inversion logic removed) that wait until both inputs are present before sending them on their respective journeys, which can be used to ensure synchronization between different parts of a circuit. This gate could instead be built purely using the other logic gates already shown above, at the cost of clarity.

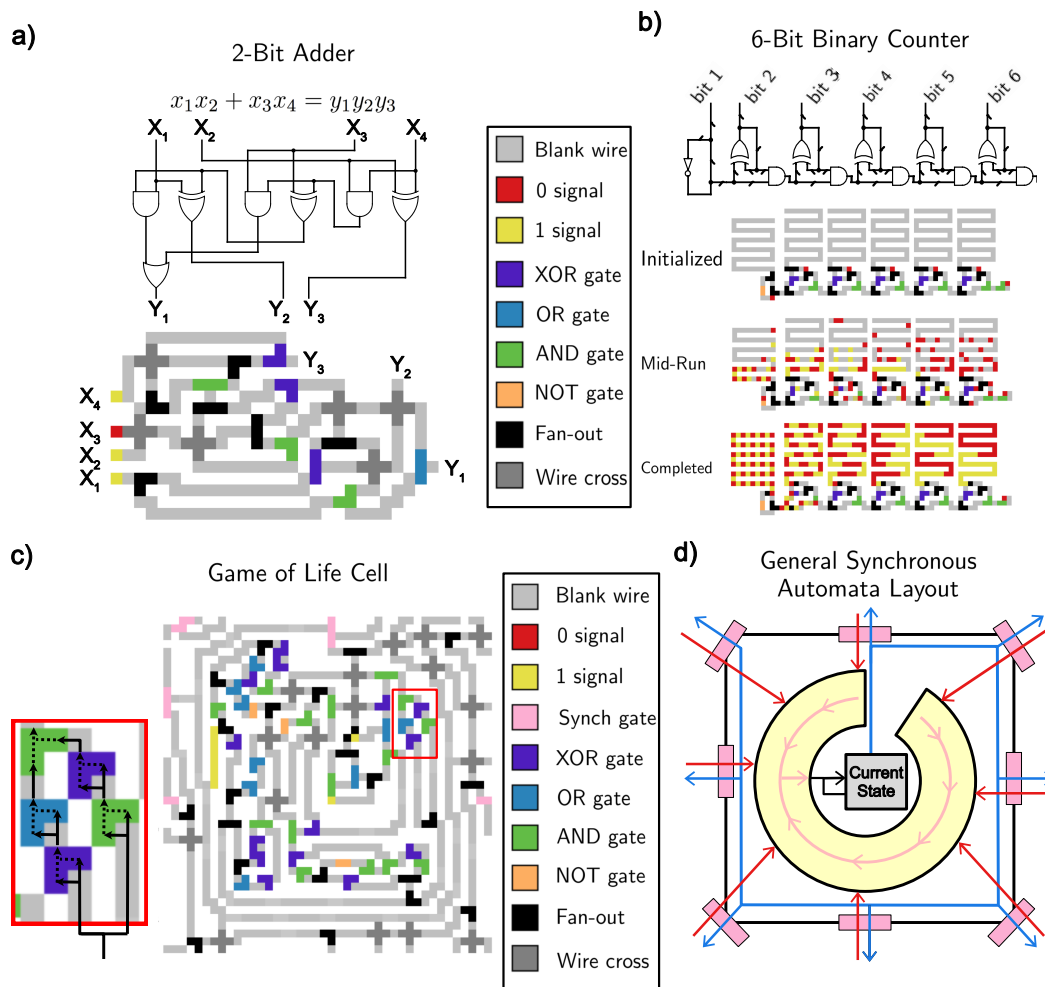


Figure 2.4: Example logic circuits and a logic circuit emulation of locally synchronous cellular automata. **a-c)** Example circuits built using the rules and gates from Figure 2.3, plus a synchronization gate. Inset in **(c)** highlights the use of “sequential gates” whose two inputs are sequential signals on the same input line, which are useful for efficiently computing certain Boolean statements with repeated structure. Arrows show the direction of logic flow through this motif, which outputs “1” if and only if exactly three out of four sequential inputs are “1”s. **d)** General scheme for building locally synchronous cellular automata using CRNs on a surface. Blue arrows represent outgoing state information, red arrows represent incoming state information, and pink arrows represent information flow through a “logic ring” that computes the next cell state. Pink rectangles are synchronization gates, which allow signals to pass only once both inputs are present.

Given these reactions and gate layouts, we can compose circuit elements and scale up to larger circuits by simply laying out signals in different initial conditions on the surface. An example surface CRN circuit that adds two 2-bit numbers is shown in Figure 2.4a. Input signals (0 or 1) are initiated at x_1, \dots, x_4 . These signals could be set manually during construction of the circuit, or they could be fed from an upstream source (perhaps another circuit). Signals asynchronously propagate through the circuit, with computations analogous to those in electronic circuits performed at each gate. The speed and exact timing of each reaction is random, but the eventual output of the circuit is guaranteed to be correct.

Another example that takes advantage of the continuous and reusable nature of logic circuits on a surface CRN is a 6-bit binary counter shown in its initial condition, after several rounds of output, and in its final state (Figure 2.4b). At the top of this circuit, a signal diffuses around a ring with a NOT gate (which latches diffusion to one direction), switching back and forth between 0 and 1. This signal is duplicated and passed through a series of sequential half-adder circuits, where each half-adder's inputs are its last computed value and the value computed by the previous half-adder. To make the counter's function clearer, a wire at the output of each half-adder stores the output history of that adder, with positions farther down the wire holding older values than positions closer to the half-adder. The output can be read as a series of binary numbers, where the n th packet (starting from the end) of the m th output line is the m th bit of the n th number. These outputs could be hooked to the input of another circuit to "clock" that circuit, or the counter could be used to create repeating spatial patterns.

Just as we can emulate logic circuits in a Game of Life, we can emulate the Game of Life using feedback logic circuits. Figure 2.4c shows a single cell from a Game of Life, implemented with a roughly 40×40 surface CRN. For scale, if such a cell were implemented on the scale of a metal crystal, with a single site at each atom, it would be order of magnitude 10 nm square; if instead it were implemented using the existing DNA origami-based scheme in (Qian and E. Winfree, 2014), with origami tiles similar to those in (P. Petersen, Tikhomirov, and Qian, 2018), it would be approximately 200 nm across, or between 1/10 to 1/5 the length of *E. coli*, with each cell's circuit layout approximately covering a $\sim 3 \times 3$ array of origami tiles.

Each cell consists of: a core loop in the center of the device that holds the cell's current state (0 or 1); an outer loop that transmits a copy of the cell's current state to each of its neighbors; and a ring-shaped block of logic that decides what the core loop's next state should be based on the values of the cell's neighbors. This basic structure, outlined in Figure 2.4d, is general for simulating any synchronous cellular automata with local update rules computable by a logic circuit, with different circuits in the logic ring yielding different automata.

At the beginning of each generation update, the state in the core loop is duplicated and sent a) to the outer loop for transmission, and b) back to the core loop for storage and computation of the next cycle state. The split signal runs clockwise around the outer ring starting from the top. At each border with another cell, a copy of the state

signal is created and waits until a matching signal is received from the neighbor. Synchronization gates (see Figure 2.3f) at these junctions ensure that no cell can update more than one step ahead of its neighbors. Once the neighbor's signal is received, it is fed into the logic ring, where a circuit in the logic ring determines what the cell's next state should be. Finally, the result of that computation is fed into the core loop, replacing the previous value, and the cycle begins again.

The Game of Life update calculation involves detecting whether or not exactly three or four out of nine neighbors are "alive" (pass a "1" signal), which is somewhat cumbersome to check using a Boolean logic circuit. To keep this circuit to a reasonable size, we take advantage of the digital, sequential nature of surface CRN logic circuits to make "sequential" logic gates that act on two sequential signals from the same input line, rather than two parallel inputs on separate input lines. For example, the inset shown in Figure 2.4c shows a five-gate motif that produces output with every four consecutive signals it receives, yielding "1" if exactly three of those signals are "1"s and "0" otherwise. An equivalent standard Boolean logic circuit would require two additional logic gates and approximately five wire crosses and four fan-outs. Note that sequential logic gates use exactly the same states and transition rules as a standard surface CRN logic gate, only arranged differently on the surface.

At first glance, this implementation of the Game of Life seems cumbersome compared to the much more compact emulation method described in Section 2.3. Using logic circuits certainly requires more surface area and more running time than directly emulating the same Game of Life. However, the circuit-based implementation uses 110 reactions as written (including synchronization gates and "repeater gates" that serve only to speed computation; see the Game of Life examples on our website for details). This number can be reduced to 46 by removing redundant gates and re-using rules for data-loading using a common input state, bringing the total reaction rule set down to less than a third the number required for the broadcast-swap-sum Game of Life emulation (see Box 5). Moreover, the same strategy can be used to construct any synchronous cellular automata using exactly the same rule set, varying only the initial layout of species on the surface. This is a considerable advantage in an experimental setting, as only one set of on-surface reactions need be designed and validated, along with the ability to lay down arbitrary initial surface layouts, in order to build any molecular-implemented (synchronous or asynchronous!) cellular automata, recurrent digital circuit, or Turing machine. In principle, even a fully functional microprocessor (Martin et al., 1989) could be implemented using exactly the same surface CRN reactions shown here. (Implementing universal circuit constructions with reversible surface CRN reactions also establishes connections to the physics of computation (Brailovskaya et al., 2019).)

This ease of scaling contrasts with existing programmable molecular implementations of logic using, for example, DNA complexes in a well-mixed test tube or protein levels in a cell. In those schemes, each data line is encoded with a different kind of molecule, and each new gate requires the design of a set of chemical

reactions with new molecular species.

Box 5: Efficient Logic Circuits

Wires, signal diffusion, 2- and 3-output fan-out, wire-crossing, synchronization and repeater gates, NOT, OR, XOR, and NOT gates—as we have demonstrated here—require a total of 110 transition rules. That is a lot of molecular interactions, from the point of view of an experimenter trying to physically build such a circuit. We can shrink this rule set considerably, sacrificing only spatial compactness and readability of the final circuit design, by removing repeater, sync, AND, OR, XOR, and NOT gates (as these can be emulated with only NOR gates), and by eliminating the 3-output fan-out gate. This brings the total number of reactions required for logic down to 46 transition rules and about as many unique states. Can you do better?

- What is the smallest rule set you can come up with that still allows you to build continuously active recurrent logic circuits?
- What is the smallest number of total unique states you can allow and still create a rule set that allows you to build continuously active recurrent logic circuits?
- Consider a surface CRN with a limited number of states available at each site. One might, for example, build a surface CRN with many different species at *different* sites, but in which each single site may only flip between N different species. Our logic circuit implementation requires up to seven species at any single site. What is the minimum number of species per site you need for logic? Can you do it with 3? With 2?
- Can you build Boolean logic circuits using only reversible reactions? (Hint: you can.) What would drive these gates forward?

Of course, the benefits of surface CRNs are not free, and this example is the first that clearly illustrates the design tradeoff alluded to at the end of Section 2.2—in this case, surface CRN-based logic circuits trade difficulty in molecular *interaction design* for difficulty of molecular *placement*. If you want to build a large logic circuit by implementing a surface CRN, you must be able to precisely position a large number of molecules into the correct initial state; in return, you can build circuits of arbitrary size with a relatively small, fixed number of molecular interaction designs.

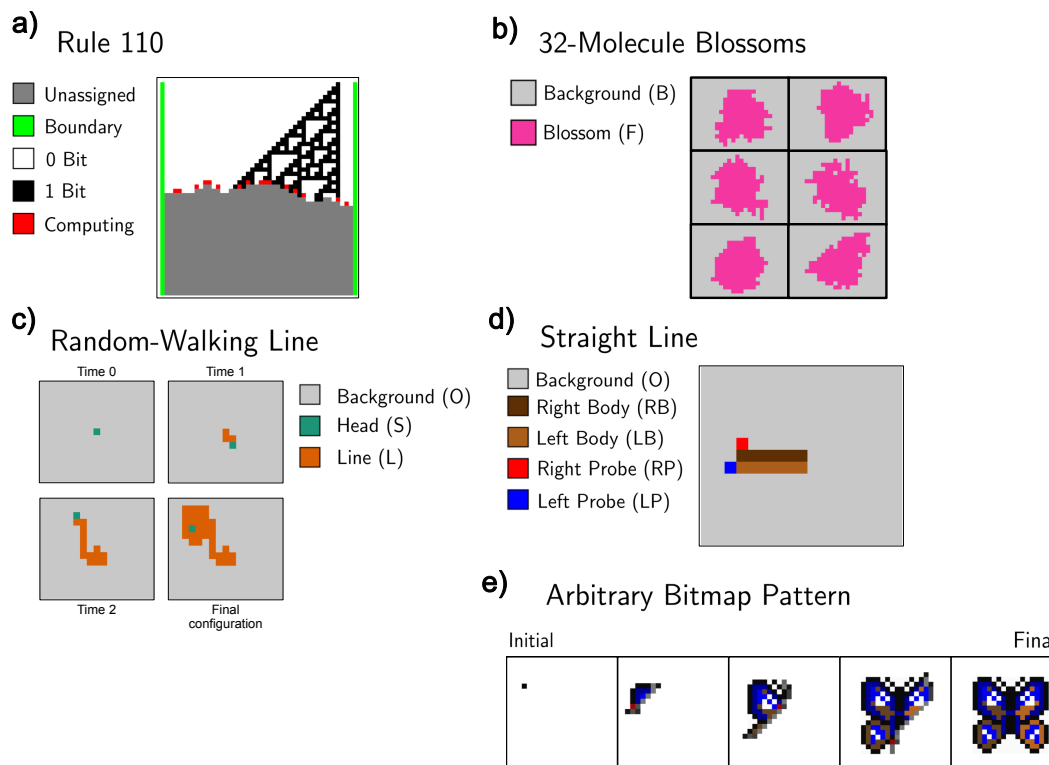


Figure 2.5: Several examples of pattern manufacture. **a)** Stephen Wolfram’s rule 110 elementary cellular automata, in mid-construction. In each row, cells use transition rules to exchange state information, then update the state of the cell below them according to rule 110. See the “Elementary Cellular Automata Rule 110 (space-time history)” example on our online simulator for details. **b)** Six final configurations of an irregular blossom of fixed size, each produced from an identical single-molecule seed on a uniform background. The exact pattern of the blossom is stochastic, but it will always have exactly 32 area. **c)** Construction of a random-walking line with the rule $S + O \rightarrow L + S$ at four times. At the final configuration, this surface CRN is stuck. **d)** Construction of a straight line. Note that in this global state, there are no cells in an RH or LH state. **e)** Construction of an arbitrary bitmap pattern from a single starting seed against a uniform background.

2.5 Manufacturing

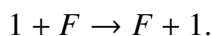
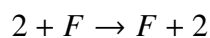
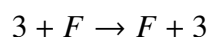
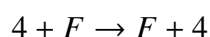
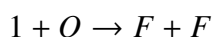
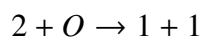
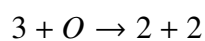
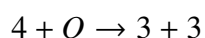
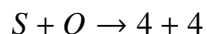
We have now seen that a major advantage of performing chemistry on a surface is that you can exploit power in the form of *spatial arrangement and placement* of molecules in order to reduce the complexity of desired *molecular interactions* for accomplishing the same task. We can also invert this paradigm to do the opposite—harness the ability to specify and implement precise and diverse *molecular interactions* in order to achieve complex *spatial arrangement and placement* from simple initial conditions.

Readers of a certain bent may have noticed that the 6-bit binary counter presented in Section 2.4 generated highly specific spatial patterns—in this case, alternating stripes of various lengths. The binary counter circuit can generate an arbitrarily long stream of patterned bits, which, given empty space to diffuse into, will create

arbitrarily large striped patterns. What other patterns can we spontaneously generate using surface CRNs? Can surface CRNs be used to manufacture complex patterns from simpler initial conditions? The answers to these questions have clear relevance to molecular manufacturing.

We have already shown that the space of surface CRNs effectively contains all cellular automata, so patterns producible by a cellular automata ought to be producible by a surface CRN as well. For example, we can make spirals with a surface CRN Greenberg-Hastings excitable media. Any of Stephen Wolfram’s one-dimensional elementary cellular automata, along with their time histories (Wolfram, 1984), can be generated using a rule set similar to, but much more compact than, the spinning-arrow class of synchronous cellular automata (Figure 2.5a). This immediately allows the construction of striping, aperiodic chaos, tree structures and Sierpiński triangles, and much more (see rules 184, 30, 90, and 110, respectively). There is a rich history of using cellular automata to model biological pattern formation and morphogenesis (Deutsch and Dormann, 2018; Mordvintsev et al., 2020) so in that sense it is natural to expect that complex and useful structures can be built by surface CRNs as well.

What about specific shapes that do not correspond easily to known automata, or that might be generated more easily with a direct surface CRN implementation? We can start with a very simple manufacturing example: construction of an irregular “blossom” of fixed, arbitrary size. We begin with a field of blank species (O), on which we add a single seed state (S). Here is a set of rules that converts the initial seed into an irregular blossom with area exactly 32 (Figure 2.5b):



The seed generates two “4” states, which each decay into two “3” states, which continue to decay in a similar fashion until a final state (F). Five rules are required to encode the splitting behavior, and four diffusion rules prevent intermediate numbered states from becoming stuck in a sea of F s. From a single seed state (say, a detection event by some molecular sensor), we obtain a large irregular shape with a defined

size (say, a large fluorescent dot visible under a microscope or by eye). With two more rules, we can double the area of the blossom; in general, the number of rules required scales with the log of the size of the blossom.

This example has the general quality of creating something large from something small and easy to initialize, but the blossoms are not particularly complex, nor particularly specific. What about a more structured structure?

One such example is an infinite straight line, again starting from a single seed S against a field of open space states O . This exercise is worth trying before you read our solution! It can be done, and relatively simply. As a warm-up exercise, note that drawing a random one-dimensional curve is almost trivial—we can do it with a single reaction $S + O \rightarrow L + S$. This curve will random-walk around the surface until it loops on itself and gets stuck (Figure 2.5c).

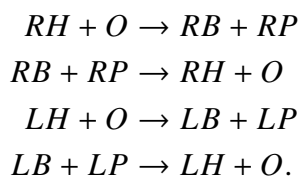
Box 6: Manufacturing Challenges

It is safe to say that the field of surface CRN manufacturing is a young one, and many challenges remain. Here are three for you to tackle:

- Our line-growing construction relies very much on an underlying square-grid geometry. It will not, for example, work as-written on a hexagonal grid. Can you design a version that does?
- Can you build a CRN that constructs, from an initial seed, a straight line of a specific length, rather than an infinite line? Can you do it in a way such that the number of rules required scales less than linearly with the length of the line? What about a square?
- Consider the Busy Beaver problem in computer science: what is the largest finite number of 1s that can be generated with a Turing Machine with a given number of states and symbols, starting with a blank tape? Similarly, we can consider the surface CRN Busy Beaver problem: given integers R and K , define $BB_{\text{CRN}}^{2\text{D}}(R, K)$ to be the largest finite, fixed number of 1s that can be reliably produced by every execution of some surface CRN with R reactions and K species, starting from a single S in a sea of O on a 2D square grid. Here, “reliably produced” means that there are a finite number of reachable states, one or more of which are terminal in the sense that no further reactions are possible, every reachable state can reach a terminal state, and every terminal state has the same number of 1s (in any arrangement, and with any other species also present). The “blossom” construction, for example, already provides a set of lower bounds $BB_{\text{CRN}}^{2\text{D}}(2n - 1, n + 2) \geq 2^n$. Can you do better? You might want to start with the similarly defined $BB_{\text{CRN}}^{1\text{D}}(R, K)$ that is confined to a 1D line of states. Or, to illuminate the influence of geometry, you might want to compare and contrast to the “zero-dimensional” well-mixed case, where $BB_{\text{CRN}}(R, K)$ concerns a stochastic CRN starting with a single S .

It is less obvious, however, how to build a *straight* line on a surface CRN. As with the spinning-arrow construction of synchronous cellular automata, here we must contend with the fact that the surface has no built-in orientation, no structure that distinguishes left from right, top from bottom, straight from sideways. Our line will have to create such a structure. One way to do this is to make the line two molecules thick, with different states on the left and right sides. A reaction, $S + O \rightarrow RH + LH$ breaks the symmetry of the background and sets the line in motion. The two head

states (reversibly) extrude probes, which look for the forward direction:



The only places where the probes can be extended such that they touch is in front of the growing line; when they are next to each other, the two probes can convert themselves into a new head with the reaction $RP + LP \rightarrow RH + LH$, which can then extrude its own probes and continue extending the line indefinitely. Note that the initial direction in which the line grows is random, determined stochastically by the first reaction and the first probe-connecting growth move, but once growth begins, it will continue indefinitely in a straight line (Figure 2.5d).

A single line extending infinitely in a random direction is admittedly not, in itself, a particularly compelling shape. However, a line can be used as a manufacturing primitive in the construction of larger, more complex shapes, like a square or triangle, which could themselves be combined into yet-more-complex shapes, perhaps eventually forming a house, or a factory. The reader might enjoy trying these and other more complex manufacturing challenges and exploring their connection to computability via the Busy Beaver problem (Rado, 1962) (see Box 6).

As an extreme example of construction-by-reaction, it is possible to extend the line-creation example to create a surface CRN that builds an arbitrary bitmap pattern around a single-site seed against an otherwise-uniform background (Figure 2.5e). In our implementation, a seed picks a random direction in which to extend a two-pixel-wide line in which each pixel has a unique address (i.e., there are unique species for each position), forming one edge of the image. A second, orthogonal two-pixel-wide line with unique addressing forms a second edge of the image. Then a series of reactions fill in the image starting from the corners, again with unique addressing. Finally, one reaction for each pixel converts the addressed site to a colored final state. This algorithm requires a fairly large number of reactions, which scale linearly with the number of painted pixels. Our construction could be easily modified to simulate tile self-assembly models (Doty, 2012), in which case more efficient encoding could be used for some patterns that are algorithmically compressible (Ma and Lombardi, 2008). The ability of surface CRNs to iteratively rewrite information in place, which is not possible in tile self-assembly, ought to allow more advanced constructions that provide more effective compression as well as self-healing capabilities, as has been seen in other cellular automaton models or morphogenesis (Mordvintsev et al., 2020).

Again, we wish to highlight the nature of the tradeoff in surface CRN design between the required number of molecular interactions and the required number of precisely-placed molecules. If you can already place molecules on a surface with high precision, then you do not need complex manufacturing algorithms of the kind

shown in this section, and you will probably want to leverage surface CRNs in other ways (perhaps the ones outlined in earlier sections). If you *cannot* place molecules on a surface with high precision, but you *can* implement many desired reactions between molecules, then you can use a surface CRN to build complex designs from simple initial conditions—even, given enough reactions, arbitrarily complex patterns from a single randomly-placed seed (Figure 2.5e). Either way, surface CRNs can be a helpful abstraction, but they perform different work depending on what technology you already have available.

2.6 Robots and Swarms

If we wish to construct complex patterns from stochastic reactions, we would do well to take inspiration from nature. For natural examples of complex construction, we need only look to ant and termite colonies. Termites, for example, begin to construct their nests using a simple set of stochastic behaviors: walk about at random; eventually pick up a mudball; wander with the mudball and eventually put it down, preferentially dropping it where there are other mudballs. A simple set of rules of this sort (usually supplemented by pheromone-laying behavior) is sufficient for a colony of termites to collectively build piles, then towers, and eventually complex nest structures (Grassé, 1959).

While recognizing that ants are not termites, out of deference to Langton's ant (Langton, 1986) and the overall superior charisma of ants over termites, we shall use a simple set of rules inspired by the nest-building behavior of termites to build rudimentary piles using a molecular “ant swarm,” as shown in Figure 2.6a. We begin with dirt species scattered randomly and uniformly on a field of open spaces. We place ants on this resource-rich field, and allow the ants to diffuse at random using rules similar to those used at the beginning of Section 2.3. Let us call this “searching.” The ants can pick up dirt, transforming into an “ant-with-dirt” species. Dirt-laden ants diffuse about randomly as well until they hit another dirt, at which point they deposit the dirt on the pile, forming a stacked-dirt species. A simple two-rule dirt physics allows any stacked dirt to diffuse around the dirt pile until it “falls off” onto an empty space (not unlike simple sandpile physics (Wiesenfeld, Tang, and Bak, 1989)). This dirt physics helps the mounds develop a more round shape. Ants which have just put down dirt enter a temporary “leaving” state in which they cannot pick up dirt, so that they diffuse slightly away from the pile they just added to before they can pick up additional dirt.

While an observer could hardly mistake the products of this series of reactions for the complex nests of real ants, our surface CRN mound-builders are at least capable of clustering scattered dirt particles into rudimentary piles. Furthermore, they do so with considerably less underlying complexity than a real ant, and are far smaller: using molecular implementations of surface CRNs along the same lines as those in (Qian and E. Winfree, 2014), our surface CRN ants act on a roughly 10^6 -fold smaller length scale, with about 10^{18} -fold less mass.

Ants, of course, do much more than build small piles of dirt. Collective behaviors

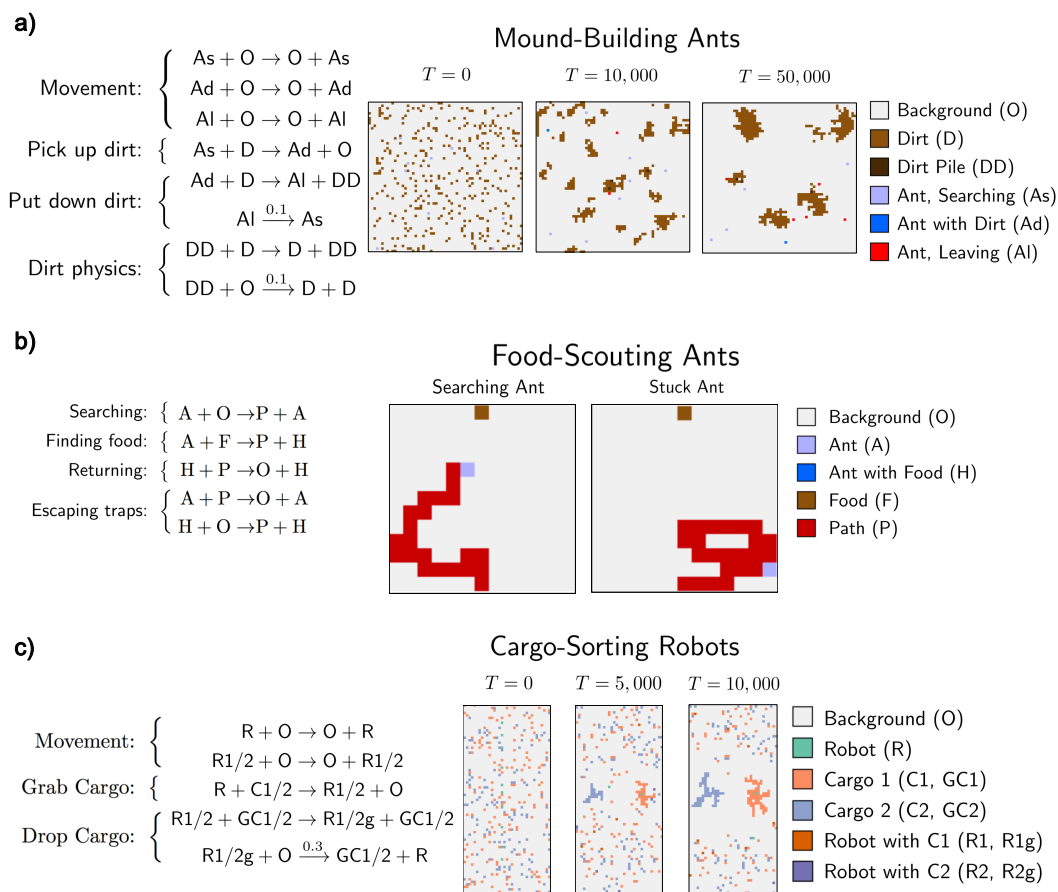


Figure 2.6: Surface CRNs with ant-like behavior. **a)** A swarm of ants (lavender, blue, red) collecting dirt (brown) into small piles. Unlabeled reactions have rates of 1/sec. **b)** An ant (purple) scouting for food (brown), leaving a trail (red) to lead itself back home when it finds the food. This ant is capable of finding food and returning to its starting position, but it is not particularly robust—it will sometimes hem itself in with its own trail, trapping itself in a ring that can be difficult to escape. **c)** A swarm of cargo-sorting robots (green) sorting two different molecular cargoes (blue and orange) into separate piles.

in animals and insects—ants included—have inspired a remarkable variety of algorithms for swarm robotics and distributed computation (Resnick, 1997; Bayındır, 2016; Dorigo and Stützle, 2019). So what other ant-like behaviors might we mimic with a surface CRN?

What about scouting for food and bringing it back to the nest? We can make an extremely simple (and extremely unintelligent) food-scouting ant (A) that diffuses about randomly, converting open spaces (O) in its wake to pheromone species P , with the reaction $A + O \rightarrow P + A$. When the ant finds food, it picks up the food and converts to a “food-bearing ant” state H with the reaction $A + F \rightarrow P + H$. A food-bearing ant walks back down its pheromone trail with the reaction $H + P \rightarrow O + H$, converting it back into open space, until it reaches its home (Figure 2.6b). This simple ant works... so long as it does not loop back on itself and become stuck. We

Molecular Rugby

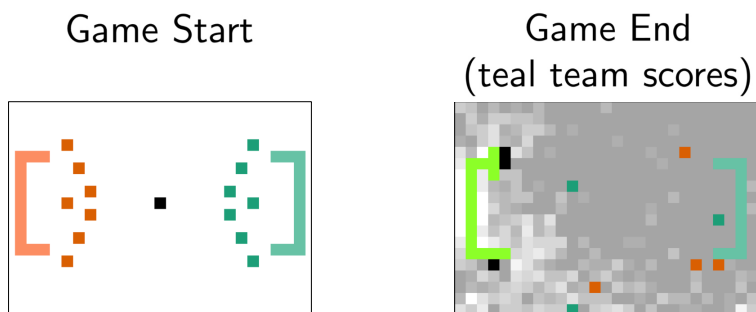


Figure 2.7: Molecular rugby. Each player submits rules defining how their team interacts with the field. Can you come up with a strategy that beats blind diffusion?

can improve the ant scout’s performance by adding “rails” to the side of its trail to prevent the trail from overlapping with itself, and by adding a very slow reaction that causes the ant to spontaneously run back to the nest, destroying its trail along the way, so that it can unstick itself from loops (see the “Genius Scouting Ant” example on our simulator webpage for implementation details). The problem of defining and implementing a better search algorithm for molecules on a surface is an open one.

The food-finding ant example bears some resemblance to an existing class of programmable surface chemistry: surface-based molecular robotics. Currently, molecular robots can traverse pre-laid tracks or open landscapes (Lund et al., 2010; Wickham et al., 2012; Muscat, Bath, and Turberfiel, 2011; Kudernac et al., 2011). Surface-based DNA robots can also sort initially-dispersed cargo into separate homogeneous piles (Thubagere et al., 2017). Simple molecular robots that do little more than walk on a DNA origami surface can take steps as often as once per second (J. Li et al., 2018), raising the hope that more complex molecular robots need not be slow. Surface CRNs are a natural model for DNA walkers and similar molecular robots, and provide a natural format for designing and testing molecular robot algorithms.

Figure 2.6c shows an algorithm for a molecular robot (R) that diffuses around a field of open sites (O) searching for two types of cargo ($C1$ and $C2$), which it carries back to two goal sites ($GC1$ and $GC2$, respectively). This algorithm bears close resemblance to the ant mound-building algorithm at the beginning of this section. The primary difference between the two is that the cargo-sorting robot distinguishes between a cargo and its goal, and can mark dropped cargo as belonging to the goal. Because these robots drop the cargo as soon as they encounter a goal, the resulting piles develop shapes similar to diffusion-limited aggregation (ThomasWitten Jr and Sander, 1981). Whether this is realistic for a molecular robot will depend on the implementation of that robot and the nature of its cargo. An alternative formulation, closer to ref. (Thubagere et al., 2017), could use specially-marked “destination” sites that are replaced by the cargo.

We may notice at this point that a common pattern with surface CRN swarm robots

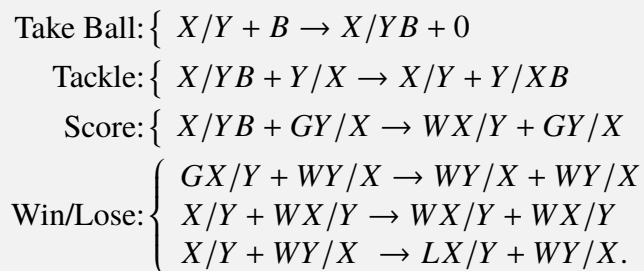
is that they all must contend with the extremely limited local information available to each species. A molecule on a surface can tell if it is next to, for example, a “dirt” molecule, but it has no way to directly measure whether that dirt is part of a larger pile, or how big that pile is—information which an ant brain can, presumably, compute from a number of nonlocal sensory cues from its eyes, limbs, olfactory system, and antennae.

Difficulties of extreme information locality can be viewed as an inherent *limitation* of surface CRN chemistry, which requires extra design effort to circumvent. Viewed another way, these difficulties are clarifying. If you are a molecule, or even an assemblage of molecules as staggeringly complex as, say, a bacteria, you do not necessarily have immediate access to “obvious” information like which direction is forward, or how big a nearby object is. These are real limitations on molecular systems, and molecular machines must either work without such information, or dedicate resources to computing it.

To explore these and other difficulties of molecular design, we propose the world’s first (to our knowledge) molecular sport—molecular rugby. In molecular rugby, states representing players on two opposing teams, X and Y , compete to pick up and move a ball to a goal protected by the other team. Fixed transition rules govern basic mechanics (picking up the ball, tackling other players, and scoring). The game is played between teams with various (constrained) transition rules controlling the movement of players on their team. See Figure 2.6 and Box 7 for details.

Box 7: Molecular Rugby

Molecular rugby is played on a fixed initial field, with a ball, seven X players and seven Y players, and a goal for each made of GX and GY states, respectively. The background of the playing field is made up of states $n \in \{0, 1, \dots, 9\}$, initialized to 0. A set of fixed rules governs the interactions between ball, players, and goals, allowing each team to pick up the ball, tackle other players to steal the ball, and score by bringing the ball to the appropriate goal:



Your challenge as a molecular rugby coach is to program interactions between your players (either X or Y) and the background, i.e., reactions of the form $X + n \rightarrow X + n'$ or $X + n \rightarrow n' + X$, so that your team has a better chance of winning. No other reactions are allowed. The fixed reactions each have rate 10, and you have a total rate budget of 200 to distribute among your team's reactions.

- We provide two team implementations (a straightforward random-walk implementation and an implementation that changes the field) on our simulator website, in the “Molecular Robot Rugby Competition” example. Can you build a ruleset that beats ours more than half the time?
- In a competition between K teams, each team plays each other team N times. If one team makes at least \sqrt{N} goals more than the other, it is declared the victor; otherwise the team with the fewer reaction rules is the victor. Host a competition among your friends!
- Invent your own variant of molecular rugby. Perhaps start with different background patterns, such as vertical stripes as yard lines. Perhaps let the background decay like pheromones using reactions $9 \rightarrow 8, 8 \rightarrow 7, \dots$. Perhaps have more or less background states. Perhaps allow team players to have more internal states, e.g., $X1, X2, \dots$. Perhaps....

2.7 Conclusions

What, in principle, can chemistry on a surface do?

We have addressed this question with a simple asynchronous cellular-automata-like framework—the surface CRN—that serves as a tractable and comprehensible model of chemistry on a surface. In this model, unimolecular and bimolecular reactions specified in a rule set can occur at any site and between any two neighboring sites on a surface, respectively. Surface CRNs may not capture the full richness of all possible physics that can occur on a surface surrounded by chemicals, but we claim that they capture many of the important features of chemistry-on-a-surface in much the same way that well-mixed CRNs capture many of the important features of chemistry-in-a-tube.

What can chemistry do on a surface, according to the surface CRN model? The short answer is that chemistry on a surface is Turing-universal, and so in principle can do anything that a computer can do.

More concretely, there are relatively simple and comprehensible surface CRN designs of reaction-diffusion behavior, direct simulation of synchronous cellular automata, and arbitrary Boolean logic circuits. We have also defined surface CRNs with simple pattern formation and simple swarm behaviors, and we believe that more complex and diverse behaviors are possible with larger rule sets involving more surface reactions.

What constraints, if any, does being on a surface impose on chemistry?

According to the surface CRN model, as we have seen from the manufacturing and swarm examples, computation with just local information and no absolute orientation requires different algorithms compared to that with both local and global information. The differences often lead to either larger numbers of reaction steps to accomplish a task or larger rule sets to compute the information that are not immediately available.

What is surface CRN better at doing than well-mixed CRNs and polymer CRNs?

In well-mixed CRNs, a larger number of specific molecular interactions is required for computing a more complex task. In practice, any compromise in that specificity would result in undesired side reactions that limit the scalability of the CRNs. Surface CRNs trade off complexity of *molecular interaction* for complexity of *spatial placement*; if you can precisely place molecules on a surface, then you can program behaviors with fewer molecular interactions, which will allow skilled chemists to build more complex chemical systems with dynamic and algorithmic behaviors.

As an example, let us compare the logic circuit designs shown in Section 2.4 with the seesaw logic circuit scheme described in (Qian and E. Winfree, 2011), which is a well-mixed CRN implemented with DNA molecules. The surface CRN implementation requires between two and three times as many reactions for a single logic gate as the seesaw implementation, but, critically, the seesaw gate implementation

requires new molecular interactions for every gate in the circuit, while the surface CRN implementation has a fixed set of rules for circuits of any size. The surface CRN's ability to reuse reactions for multiple logic gates comes directly from its ability to spatially separate components—it replaces chemical specification with spatial separation, which for many systems should prove easier to achieve. Because of the fixed set of rules for arbitrary circuits, undesired side reactions do not scale with the complexity of the surface CRN. Unlike the well-mixed CRN, it is not necessary to reduce the side reactions by reducing concentrations of chemical species in the surface CRN, allowing each reaction in the surface CRN to be as fast as possible regardless of the circuit size.

Moreover, surface CRNs can be massively parallel. Billions to trillions of tiny surfaces could float around in one test tube, each carrying out a distinct computation. In contrast, a well-mixed CRN in one test tube can compute exactly one thing at any time. In principle, polymer CRNs can be parallel (Bennett, 1982), but existing constructions of polymer CRNs using DNA nanotechnology require a single copy of certain polymers in order to be Turing universal (Qian, Soloveichik, and E. Winfree, 2011; Lakin and Phillips, 2011; Tai and Condon, 2019), severely limiting their capability for parallel computation.

The theoretical understanding that we have explored in this work will hopefully inspire construction of increasingly interesting chemical systems on surfaces. There already exist systematic methods for implementing arbitrary well-mixed CRNs using DNA nanotechnology, both in theory (Soloveichik, Seelig, and E. Winfree, 2010) and practice (Chen, Dalchau, et al., 2013; Srinivas et al., 2017), as well a theoretical approach for implementing arbitrary surface CRNs using DNA strand displacement reactions attached to a DNA origami surface (Qian and E. Winfree, 2014). Thus, the surface CRN is more than a theoretical tool for understanding the abstract properties of surface chemistry—it can, in principle, be physically implemented in a real-world laboratory. These implementations will provide a starting point for the development of complex, programmed molecular systems that may someday be extended to other types of molecules including RNA and proteins.

Besides the engineering effort on molecular machines, studies on the surface CRN model will also help answer a fundamental question: what capabilities and constraints of molecular interactions guided the origin and evolution of life? While surface CRNs are more scalable and parallel than well-mixed and polymer CRNs, no single type of chemistry alone provides the sole solution for life-level complexity. The receptors on cell membranes represent an example of surface chemistry. The filaments in cytoskeletons represent an example of polymer chemistry. The transcription factors in genetic regulatory circuits represent an example of well-mixed chemistry. Understanding the advantages and tradeoffs of each type of chemistry will clarify the design principles for both engineered and natural molecular systems, for example defining what the best geometry (or combination of geometries) is for solving a given molecular task. Naturally, this will require investigation of more complex possibilities, including three-dimensional geometry, reconfiguration be-

tween different geometries, and integration of different geometries, and will put us on the road toward understanding and engineering molecular systems with behaviors as sophisticated as those seen in biology.

References

- Ballegooijen, Marijn van and Maarten Boerlijst (2004). “Emergent trade-offs and selection for outbreak frequency in spatial epidemics”. In: *Proceedings of the National Academy of Sciences* 101, pp. 18246–18250.
- Bathe, Mark and Paul Rothemund (2017). “DNA nanotechnology: A foundation for programmable nanoscale materials”. In: *MRS Bulletin* 42, pp. 882–888.
- Bayindir, Levent (2016). “A review of swarm robotics tasks”. In: *Neurocomputing* 172, pp. 292–321.
- Bennett, Charles (1982). “The thermodynamics of computation—a review”. In: *International Journal of Theoretical Physics* 21, pp. 905–940.
- Brailovskaya, Tatiana, Gokul Gowri, Sean Yu, and Erik Winfree (2019). “Reversible computation using swap reactions on a surface”. In: *DNA Computing and Molecular Programming (Lecture Notes in Computer Science)* 11648, pp. 174–196.
- Bray, William (1921). “A periodic reaction in homogeneous solution and its relation to catalysis”. In: *Journal of the American Chemical Society* 43, pp. 1262–1267.
- Briggs, Thomas and Warren Rauscher (1973). “An oscillating iodine clock”. In: *Journal of Chemical Education* 50, p. 496.
- Bui, Hieu, Shalin Shah, Reem Mokhtar, Tianqi Song, Sudhanshu Garg, and John Reif (2018). “Localized DNA hybridization chain reactions on DNA origami”. In: *ACS Nano* 12, pp. 1146–1155.
- Cappelletti, Daniele, Andrés Ortiz-Muñoz, David F Anderson, and Erik Winfree (2020). “Stochastic chemical reaction networks for robustly approximating arbitrary probability distributions”. In: *Theoretical Computer Science* 801, pp. 64–95.
- Chandran, Harish, Nikhil Gopalkrishnan, Andrew Phillips, and John Reif (2011). “Localized hybridization circuits”. In: *DNA Computing and Molecular Programming (Lecture Notes in Computer Science)* 6937, pp. 64–83.
- Chatterjee, Gourab, Neil Dalchau, Richard A Muscat, Andrew Phillips, and Georg Seelig (2017). “A spatially localized architecture for fast and modular DNA computing”. In: *Nature Nanotechnology* 12, pp. 920–927.
- Chen, Yuan-Jyue, Neil Dalchau, Niranjan Srinivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig (2013). “Programmable chemical controllers made from DNA”. In: *Nature Nanotechnology* 8, pp. 755–762.

- Chen, Yuan-Jyue, Benjamin Groves, Richard Muscat, and Georg Seelig (2015). “DNA nanotechnology from the test tube to the cell”. In: *Nature Nanotechnology* 10, pp. 748–760.
- Cherry, Kevin and Lulu Qian (2018). “Scaling up molecular pattern recognition with DNA-based winner-take-all neural networks”. In: *Nature* 559, pp. 370–376.
- Cook, Matthew, David Soloveichik, Erik Winfree, and Jehoshua Bruck (2009). “Programmability of chemical reaction networks”. In: *Algorithmic Bioprocesses*, pp. 543–584.
- Cummings, Rachel, David Doty, and David Soloveichik (2016). “Probability 1 computation with chemical reaction networks”. In: *Natural Computing* 15, pp. 245–261.
- Degn, Hans (1967a). “Effect of bromine derivatives of malonic acid on the oscillating reaction of malonic acid, cerium ions and bromate.” In: *Nature* 213, pp. 589–590.
- (1967b). “Evidence of a branched chain reaction in the oscillating reaction of hydrogen peroxide, iodine, and iodate”. In: *Acta Chemica Scandinavica* 21, pp. 1057–1066.
- Deutsch, Andreas and Sabine Dormann (2018). *Cellular automaton modeling of biological pattern formation, 2nd edition*. Birkhäuser.
- Dorigo, Marco and Thomas Stützle (2019). “Ant colony optimization: overview and recent advances”. In: *Handbook of Metaheuristics*, pp. 311–351.
- Doty, David (2012). “Theory of algorithmic self-assembly”. In: *Communications of the ACM* 55, pp. 78–88.
- Douglas, Shawn, Hendrik Dietz, Tim Liedl, Björn Högberg, Franziska Graf, and William Shih (2009). “Self-assembly of DNA into nanoscale three-dimensional shapes”. In: *Nature* 459, pp. 414–418.
- Fages, François, Guillaume Le Guludec, Olivier Bournez, and Amaury Pouly (2017). “Strong Turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs”. In: *Lecture Notes in Computer Science* 10545, pp. 108–127.
- Fange, David, Otto Berg, Paul Sjöberg, and Johan Elf. (2010). “Stochastic reaction-diffusion kinetics in the microscopic limit”. In: *Proceedings of the National Academy of Sciences* 107, pp. 19820–19825.
- Field, Richard, Endre Koros, and Richard Noyes (1972). “Oscillations in chemical systems. II. Thorough analysis of temporal oscillation in the bromate—cerium—malonic acid system”. In: *Journal of the American Chemical Society* 94, pp. 8649–8664.
- Gács, Peter (2001). “Deterministic computations whose history is independent of the order of asynchronous updating”. In: *arXiv preprint*, cs/0101026.

- Gibson, Michael and Jehoshua Bruck (2000). “Efficient exact stochastic simulation of chemical systems with many species and many channels”. In: *The Journal of Physical Chemistry A* 104, pp. 1876–1889.
- Gillespie, Daniel T. (1977). “Exact stochastic simulation of coupled chemical reactions”. In: *Journal of Physical Chemistry* 81.25, pp. 2340–2361. DOI: 10.1021/j100540a008.
- Good, Matthew, Jesse Zalatan, and Wendell Lim (2011). “Scaffold proteins: hubs for controlling the flow of cellular information”. In: *Science* 332, pp. 680–686.
- Grassberger, Peter and Holger Kantz (1991). “On a forest fire model with supposed self-organized criticality”. In: *Journal of Statistical Physics* 63, pp. 685–700.
- Grassé, Pierre-Paul (1959). “La reconstruction du nid et les coordinations interindividuelles chez *Bellicositermes natalensis* et *Cubitermes sp.* la théorie de la stigmergie: Essai d’interprétation du comportement des termites constructeurs”. In: *Insectes Sociaux* 6, pp. 41–80.
- Greenberg, James, Curtis Greene, and Stuart Hastings (1980). “A combinatorial problem arising in the study of reaction-diffusion equations”. In: *SIAM Journal on Algebraic Discrete Methods* 1, pp. 34–42.
- Greenberg, James and Stuart Hastings (1978). “Spatial patterns for discrete models of diffusion in excitable media”. In: *SIAM Journal on Applied Mathematics* 34, pp. 515–523.
- Hattne, Johan, David Fange, and Johan Elf (2005). “Stochastic reaction-diffusion simulation with MesoRD”. In: *Bioinformatics* 21, pp. 2923–2924.
- Ingerson, Thomas and Raymond Buvel (1984). “Structure in asynchronous cellular automata”. In: *Physica D: Nonlinear Phenomena* 10, pp. 59–68.
- Jakubith, Sven, Harm-Hinrich Rotermund, Wilfried Engel, Alexander Von Oertzen, and Gerhard Ertl (1990). “Spatiotemporal concentration patterns in a surface reaction: Propagating and standing waves, rotating spirals, and turbulence”. In: *Physical Review Letters* 65, pp. 3013–3016.
- Ke, Yonggang, Luvena Ong, William Shih, and Peng Yin (2012). “Three-dimensional structures self-assembled from DNA bricks”. In: *Science* 338, pp. 1177–1183.
- Kudernac, Tibor, Nopporn Ruangsapapichat, Manfred Parschau, Beatriz Maciá, Nathalie Katsonis, Syuzanna Harutyunyan, Karl-Heinz Ernst, and Ben Feringa (2011). “Electrically driven directional motion of a four-wheeled molecule on a metal surface”. In: *Nature* 479, pp. 208–211.
- Lakin, Matthew, Rasmus Petersen, Kathryn, Gray, and Andrew Phillips (2014). “Abstract modelling of tethered DNA circuits”. In: *DNA Computing and Molecular Programming (Lecture Notes in Computer Science)* 8727, pp. 132–147.

- Lakin, Matthew and Andrew Phillips (2011). “Modelling, simulating and verifying Turing-powerful strand displacement systems”. In: *DNA Computing and Molecular Programming (Lecture Notes in Computer Science)* 6937, pp. 130–144.
- Langton, Christopher (1986). “Studying artificial life with cellular automata”. In: *Physica D: Nonlinear Phenomena* 22, pp. 120–149.
- Lee, Jia, Susumu Adachi, Ferdinand Peper, and Shinro Mashiko (2005). “Delay-insensitive computation in asynchronous cellular automata”. In: *Journal of Computer and System Sciences* 70, pp. 201–220.
- Li, Chao, Na Li, Liwei Liu, Yajie Zhang, Chenyang Yuan, Lianmao Peng, Shimin Hou, and Yongfeng Wang (2017). “Kinetically controlled hierarchical self-assemblies of all- *trans*-retinoic acid on Au (111)”. In: *Chemical Communications* 53, pp. 2252–2255.
- Li, Jieming, Alexander Johnson-Buck, Yuhe Renee Yang, William Shih, Hao Yan, and Nils Walter (2018). “Exploring the speed limit of toehold exchange with a cartwheeling DNA acrobat”. In: *Nature Nanotechnology* 13, pp. 723–729.
- Li, Na, Gaochen Gu, Xue Zhang, Daoliang Song, Yajie Zhang, Boon Teo, Lian-Mao Peng, Shimin Hou, and Yongfeng Wang (2017). “Packing fractal Sierpiński triangles into one-dimensional crystals *via* a templating method”. In: *Chemical Communications* 53, pp. 3469–3472.
- Lund, Kyle, Anthony Manzo, Nadine Dabby, Nicole Michelotti, Alexander Johnson-Buck, Jeanette Nangreave, Steven Taylor, Renjun Pei, Milan Stojanovic, Nils Walter, Erik Winfree, and Hao Yan (2010). “Molecular robots guided by prescriptive landscapes”. In: *Nature* 465, pp. 206–210.
- Ma, Xiaojun and Fabrizio Lombardi (2008). “Synthesis of tile sets for DNA self-assembly”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, pp. 963–967.
- MacLennan, Bruce (2015). “The morphogenetic path to programmable matter”. In: *Proceedings of the IEEE* 103, pp. 1226–1232.
- Martin, Alain, Steven Burns, Tak-Kwan Lee, Drazen Borkovic, and Pieter Hazewindus (1989). “The design of an asynchronous microprocessor”. In: *Decennial Caltech Conference on VLSI*, pp. 351–373.
- Mordvintsev, Alexander, Ettore Randazzo, Eyvind Niklasson, and Michael Levin (2020). “Growing Neural Cellular Automata”. In: *Distill* 5, e23.
- Muscat, Richard, Jonathan Bath, and Andrew Turberfiel (2011). “A programmable molecular robot”. In: *Nano Letters* 11, pp. 982–987.
- Muscat, Richard, Karin Strauss, Luis Ceze, and Georg Seelig (2013). “DNA-based molecular architecture with spatially localized components”. In: *ACM SIGARCH Computer Architecture News* 41, pp. 177–188.

- Nakamura, Katsuhiko (1981). “Synchronous to asynchronous transformation of polyautomata”. In: *Journal of Computer and System Sciences* 23, pp. 22–37.
- Petersen, Philip, Grigory Tikhomirov, and Lulu Qian (2018). “Information-based autonomous reconfiguration in systems of interacting DNA nanostructures”. In: *Nature Communications* 9, p. 5362.
- Poole, William, Andrés Ortiz-Muñoz, Abhishek Behera, Nick S Jones, Thomas E Ouldridge, Erik Winfree, and Manoj Gopalkrishnan (2017). “Chemical Boltzmann machines”. In: *DNA Computing and Molecular Programming (Lecture Notes in Computer Science)* 10467, pp. 210–231.
- Poundstone, William (1985). *The recursive universe: cosmic complexity and the limits of scientific knowledge*. McGraw-Hill.
- Qian, Lulu, David Soloveichik, and Erik Winfree (2011). “Efficient Turing-universal computation with DNA polymers”. In: *DNA Computing and Molecular Programming (Lecture Notes in Computer Science)* 6518, pp. 123–140.
- Qian, Lulu and Erik Winfree (2011). “Scaling up digital circuit computation with DNA strand displacement cascades”. In: *Science* 332, pp. 1196–1201.
- (2014). “Parallel and scalable computation and spatial dynamics with DNA-based chemical reaction networks on a surface”. In: *DNA Computing and Molecular Programming (Lecture Notes in Computer Science)* 8727, pp. 114–131.
- Qian, Lulu, Erik Winfree, and Jehoshua Bruck (2011). “Neural network computation with DNA strand displacement cascades”. In: *Nature* 475, pp. 368–372.
- Rado, Tibor (1962). “On non-computable functions”. In: *Bell System Technical Journal* 41, pp. 877–884.
- Rendell, Paul (2002). “Turing universality of the Game of Life”. In: *Collision-Based Computing*, pp. 513–539.
- Rennard, Jean-Philippe (2002). “Implementation of logical functions in the Game of Life”. In: *Collision-Based Computing*, pp. 491–512.
- Resnick, Mitchel (1997). *Turtles, termites, and traffic jams: Explorations in massively parallel microworlds*. MIT Press.
- Rothemund, Paul (2006). “Folding DNA to create nanoscale shapes and patterns”. In: *Nature* 440, pp. 297–302.
- Ruiz, Ismael Mullor, Jean-Michel Arbona, Amitkumar Lad, Oscar Mendoza, Jean-Pierre Aimé, and Juan Elezgaray (2015). “Connecting localized DNA strand displacement reactions”. In: *Nanoscale* 7, pp. 12970–12978.
- Scalise, Dominic and Rebecca Schulman (2019). “Controlling matter at the molecular scale with DNA circuits”. In: *Annual Review of Biomedical Engineering* 21, pp. 469–493.

- Schöneberg, Johannes and Frank Noé (2013). “ReaDDy – a software for particle-based reaction-diffusion dynamics in crowded cellular environments”. In: *PLoS One* 8, e74261.
- Schönfisch, Birgitt and André de Roos (1999). “Synchronous and asynchronous updating in cellular automata”. In: *BioSystems* 51, pp. 123–143.
- Seelig, Georg, David Soloveichik, David Yu Zhang, and Erik Winfree (2006). “Enzyme-free nucleic acid logic circuits”. In: *Science* 314, pp. 1585–1588.
- Setvin, Martin, Michele Retliccioli, Flora Poelzleitner, Jan Hulva, Michael Schmid, Lynn A Boatner, Cesare Franchini, and Ulrike Diebold (2018). “Polarity compensation mechanisms on the perovskite surface KTaO_3 (001)”. In: *Science* 359, pp. 572–575.
- Shang, Jian, Yongfeng Wang, Min Chen, Jingxin Dai, Xiong Zhou, Julian Kuttner, Gerhard Hilt, Xiang Shao, J. Michael Gottfried, and Kai Wu (2015). “Assembling molecular Sierpiński triangle fractals”. In: *Nature Chemistry* 7, pp. 389–393.
- Soloveichik, David, Matthew Cook, Erik Winfree, and Jehoshua Bruck (2008). “Computation with finite stochastic chemical reaction networks”. In: *Natural Computing* 7, pp. 615–633.
- Soloveichik, David, Georg Seelig, and Erik Winfree (2010). “DNA as a universal substrate for chemical kinetics”. In: *Proceedings of the National Academy of Sciences* 107, pp. 5393–5398.
- Srinivas, Niranjan, James Parkin, Georg Seelig, Erik Winfree, and David Soloveichik (2017). “Enzyme-free nucleic acid dynamical systems”. In: *Science* 358, eaal2052.
- Stojanovic, Milan, Stanka Semova, Dmitry Kolpashchikov, Joanne Macdonald, Clint Morgan, and Darko Stefanovic (2005). “Deoxyribozyme-based ligase logic gates and their initial circuits”. In: *Journal of the American Chemical Society* 127, pp. 6914–6915.
- Tai, Allison and Anne Condon (2019). “Error-free stable computation with polymer-supplemented chemical reaction networks”. In: *DNA Computing and Molecular Programming (Lecture Notes in Computer Science)* 11648, pp. 197–218.
- Tan, Tzer Han, Jinghui Liu, Pearson Miller, Melis Tekant, Jörn Dunkel, and Nikta Fakhri (2020). “Topological turbulence in the membrane of a living cell”. In: *Nature Physics*. DOI: 10.1038/s41567-020-0841-9.
- Teichmann, Mario, Enzo Kopperger, and Friedrich Simmel (2014). “Robustness of localized DNA strand displacement cascades”. In: *ACS Nano* 8, pp. 8487–8496.
- Thomas Witten Jr and Leonard Sander (1981). “Diffusion-limited aggregation, a kinetic critical phenomenon”. In: *Physical Review Letters* 47, pp. 1400–1403.

- Thubagere, Anupama, Wei Li, Robert Johnson, Zibo Chen, Shayan Doroudi, Yae Lim Lee, Gregory Izatt, Sarah Wittman, Niranjan Srinivas, Damien Woods, Erik Winfree, and Lulu Qian (2017). “A cargo-sorting DNA robot”. In: *Science* 357, ean6558.
- Toffoli, Tommaso and Norman Margolus (1987). “Cellular automata machines: a new environment for modeling”. In:
- Wang, Yifan, Na Xue, Ruoning Li, Tianhao Wu, Na Li, Shimin Hou, and Yongfeng Wang (2019). “Construction and properties of Sierpiński triangular fractals on surfaces”. In: *Chem Phys Chem* 20, pp. 2262–2270.
- Wickham, Shelley, Jonathan Bath, Yousuke Katsuda, Masayuki Endo, Kumi Hidaka, Hiroshi Sugiyama, and Andrew Turberfield (2012). “A DNA-based molecular motor that can navigate a network of tracks”. In: *Nature Nanotechnology* 7, pp. 169–173.
- Wiesenfeld, Kurt, Chao Tang, and Per Bak (1989). “A physicist’s sandbox”. In: *Journal of Statistical Physics* 54, pp. 1441–1458.
- Winfree, Arthur (1984). “The prehistory of the Belousov-Zhabotinsky oscillator”. In: *Journal of Chemical Education* 61, pp. 661–663.
- (2001). *The Geometry of Biological Time, Second Edition*. Springer.
- Winfree, Erik (2019). “Chemical reaction networks and stochastic local search”. In: *DNA Computing and Molecular Programming (Lecture Notes in Computer Science)* 11648, pp. 1–20.
- Wolfram, Stephen (1984). “Cellular automata as models of complexity”. In: *Nature* 311, pp. 419–424.
- Zhabotinskii, Anatol (1964). “Periodic course of the malonic acid in a solution (studies on the kinetics of Belousov’s reaction)”. In: *Biofizika* 9, pp. 306–311.
- Zhang, David Yu and Georg Seelig (2011). “Dynamic DNA nanotechnology using strand-displacement reactions”. In: *Nature Chemistry* 3, pp. 103–113.

MODELING DYNAMIC TRANSCRIPTIONAL CRISPRi CIRCUITS

3.1 Introduction

A central challenge of modern bioengineering is that of “programming” cells with complex, dynamic behaviors. Simple examples of genetically encoded dynamic functions include cell state oscillation (Elowitz and Leibler, 2000; Stricker et al., 2008; Swaminathan et al., 2016), event detection and logging (Hsiao et al., 2016), molecular fold-change detection (Goentoro et al., 2009; Kim et al., 2014), and signal level discrimination (Rubens, Selvaggio, and Lu, 2016). A common challenge when engineering complex behavior is the need for numerous *specific* interactions between components. In general, engineering specific, efficient, non-promiscuous reactions from scratch is difficult for a number of reasons, so bioengineers typically use natural systems whose components have built-in specificity and selectivity.

One such natural molecular system is that of the gene regulatory network. Synthetic gene regulatory networks exploit the ability of transcription factors to specifically control the actions of target promoters to “wire” together transcriptional units, much the same way microchip manufacturers use spatial arrangements to wire together silicon-based components like transistors and logic gates. Gene regulatory networks have been successfully used to build small circuits (Elowitz and Leibler, 2000; Gardner, Cantor, and J. J. Collins, 2000; A. a. K. Nielsen et al., 2016), but they have not been used to build systems with more than about a dozen regulators. Major barriers to scaling up genetic regulatory networks include a lack of orthogonal transcription factors (the largest verified-orthogonal library of repressors currently consists of about 16 genes (A. A. Nielsen and C. A. Voigt, 2014)), mismatches in output and input levels between different regulators, and metabolic burden on the host cell (Ceroni et al., 2015).

One system that promises to scale better than classical genetic regulators is CRISPRi, a system of repression that uses a catalytically inactive mutant of the programmable endonuclease Cas9 (“dCas”). A dCas protein is inactive until loaded with a guide RNA (gRNA) containing a roughly 20-bp variable region. Once loaded, dCas will bind to any double-stranded DNA sequence matching the variable region of the gRNA, so long as it is immediately upstream of a short PAM region (NGG for the commonly-used *S. pyogenes* dCas9, but different for different dCas variants) (Esvelt et al., 2013). Binding of dCas can interfere with prokaryotic transcription, either preventing initiation of transcription (if the gRNA is targeted within or immediately around a promoter) or blocking elongation (if the gRNA is targeted downstream of a promoter) (Qi et al., 2013). Other dCas-based transcription factors, collectively known as CRISPRa transcription factors, activate instead of repress by using native

polymerase-recruiting factors fused to dCas9 or recruited by a binding domain on the gRNA (Bikard et al., 2013; Mali et al., 2013; Gilbert et al., 2013; Chavez et al., 2016; Dong et al., 2018).

CRISPRi repressors (and CRISPRa activators) have several potential advantages over traditional transcription factors. The clearest advantage of CRISPRi is that it provides an almost limitless supply of orthogonal repressors. Another advantage of CRISPRi is the relative uniformity of CRISPRi repressors. Since many CRISPRi operators can be made using the same core promoter sequence, it might be expected that different CRISPRi repressors should act with similar input/output relationships.

Since it was first proposed (Jinek et al., 2012), CRISPRi has been widely used for biophysical characterization of Cas9 (Ma et al., 2016; Mekler et al., 2016; Singh et al., 2016; Boyle et al., 2017; Gong et al., 2017; Jones et al., 2017) and for control of host gene expression (Zalatan et al., 2015; Nissim et al., 2014; Gilbert et al., 2013). More rarely, CRISPRi has been used as a synthetic tool in eukaryotic systems. Layerable CRISPRi endpoint logic gates have been designed at least twice (A. A. Nielsen and C. A. Voigt, 2014; Gander et al., 2017), and circuits up to eight gates deep and utilizing up to a dozen gates in total have been constructed (although signal degradation over multiple layers has been a consistent problem). CRISPRi has also been used to make circuits with simple dynamic behavior (Santos-Moreno, Taisiudi, et al., 2020; Kuo et al., 2020) but has not yet been widely adapted to create scalable prokaryotic circuits.

We use a simple model to demonstrate that CRISPRi can be used to build both toggle switches and repressilators, despite the fact that CRISPRi displays no cooperativity. We predict that the CRISPRi repressilator should function under physiological conditions in *E. coli*, but only barely (an observation backed up by other experiments in the literature), and suggest several interventions that should make the CRISPRi repressilator more robust. Finally, we compare the performance of CRISPRi and CRISPRa under increasingly harsh gRNA competition for dCas enzyme, and find that CRISPRa is much more robust to gRNA competition than CRISPRi.

3.2 A Model of CRISPRi

Most of our analyses will use a mass-action ODE model, even though there is good reason to believe that at least some components of a CRISPRi network will be present at low concentrations (< 10 molecules/cell), on the grounds that 1) ODE models are easier to write, simulate, understand, and analyze than stochastic models, and 2) ODE models can provide insight even in systems where the bulk assumptions of a mass-action may not be justified. See Chapter 4 for stochastic modeling of one of our example circuits.

Our model describes dCas activity at the level of explicit binding and unbinding. We do not use a Hill function approximation, on the grounds that dCas binding and unbinding is relatively slow compared to typical timescales of biocircuit activity, and so we cannot assume that dCas will come to equilibrium quickly.

A description of the model

A simple CRISPRi model, as shown in Figure 3.1, includes the following processes:

- Production of dCas;
- Production of gRNAs from free promoters;
- (Optional) Leak production of gRNAs from dCas-bound promoters;
- Active degradation of free gRNAs (by RNAses);
- (Optional) Active degradation of dCas and dCas complexes (in addition to global dilution, e.g., by proteases);
- Global dilution;
- Maintenance of promoter copy number, and unbinding of gRNA:dCas complexes triggered by replication;
- Binding and unbinding of gRNAs from dCas;
- Binding and unbinding of gRNA:dCas complexes from target promoters;

Production of dCas is taken to be constant; gRNA production is constant from unbound promoters, and constant from bound promoters with a lower rate (possibly zero). Guide RNAs (and, optionally, dCas and dCas complexes) are actively degraded at a rate proportional to their abundance. Binding reactions (and unbinding of gRNA from dCas) follow standard mass action binding and unbinding kinetics. Except where explicitly stated otherwise, all CRISPRi promoters are assumed to have identical dynamics (aside from the identity of their repressors). This model can be modified for CRISPRa by changing gRNA production to only occur from dCas-bound promoters, instead of from free promoters, again with optional leak from unbound promoters.

Unbinding of dCas from its target promoters is a little unusual in our model. dCas binds extremely tightly to its targets. In bacterial cells, the rate of dCas unbinding from DNA is substantially slower than the rate of bacterial replication, even in non-lab-adapted strains with division times of over 100 minutes (Jones et al., 2017) (though possibly not (Gong et al., 2017)). This means that dCas effectively only unbinds as a consequence of DNA replication, with the bacterial replication machinery forcing dCas from its target. Any CRISPRi circuit with complex, non-monotonic dynamics will require either dilution, dCas degradation, or some mechanism that actively unbinds dCas from its DNA targets.

Accordingly, all models in this report apply global cell dilution to all components (including DNA, dCas, and complexes of the two), using a series of reactions of the form $X \rightarrow \emptyset$, where X is any species in the model. A caveat is that total promoter concentrations need be held constant, as dilution of DNA is assumed to

be equally balanced by replication. We implement this with a reaction $DNA_{ij} \rightarrow DNA_{ij} + DNA_{ij}$ at the rate of dilution for each DNA_{ij} in the model, where DNA_{ij} is any unbound promoter for gRNA i that is repressed by gRNA j . DNA bound to dCas follows a combined replication/unbinding reaction $DNA_{ij} : dCas_j \rightarrow DNA_{ij} + dCas_j$, again at the rate of dilution, where $dCas_j$ is a dCas molecule complexed to gRNA j . This reaction represents the current understanding that dCas is removed from DNA by DNA replication. These approximations of DNA replication will be sufficient for now, but will need to be changed when we move to stochastic simulations in Chapter 4.

Note that the reactions $DNA_{ij} \rightarrow DNA_{ij} + DNA_{ij}$ and $DNA_{ij} \rightarrow \emptyset$ make DNA_{ij} only neutrally stable – they do not reject disturbances to DNA levels. This will need to be addressed in Chapter 4, when we simulate a CRISPRiator circuit stochastically, but for the deterministic simulations we will use in this chapter, the precarious balance this simple replication mechanism achieves will be sufficient.

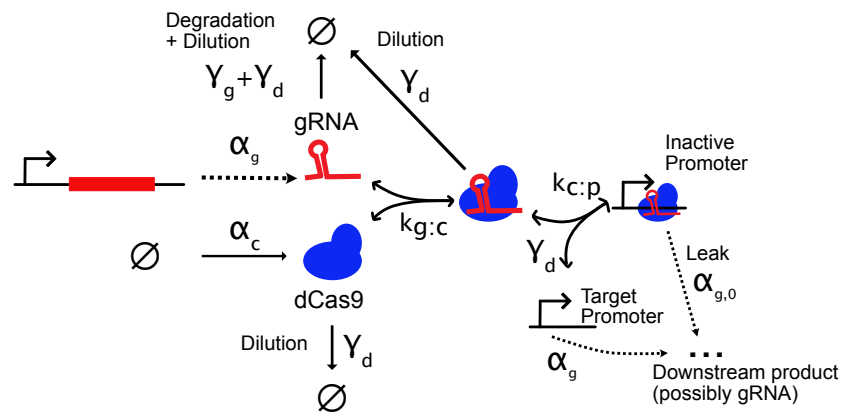
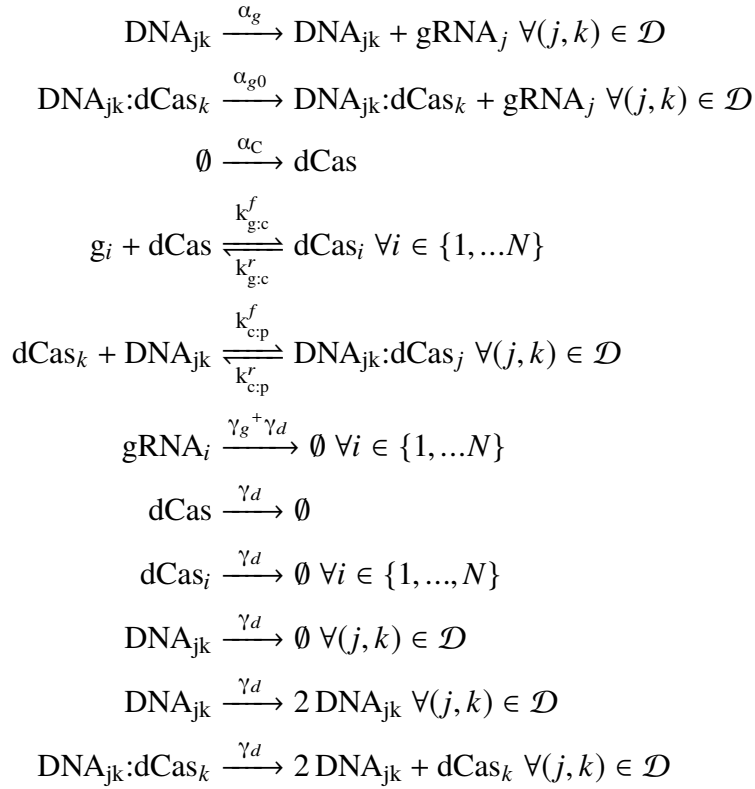


Figure 3.1: Diagrammatic representation of the CRISPRi model used in this report. Rates for active degradation of dCas and leak activity of dCas-bound promoters are set to zero in some simulations.

The full model consists of the following reactions, for a set of N expressed gRNAs $\mathcal{G} = \{gRNA_i, i = 1, \dots, N\}$ and a set of indices \mathcal{D} such that every $(j, k) \in \mathcal{D}$ corresponds to a promoter DNA_{jk} in the network that produces $gRNA_j$ and is repressed by $gRNA_k$ (where k is possibly null):



Parameterization of the model

A set of typical parameters used in simulations below is given in Table 3.1. DNA concentrations will be assumed to be 2 nM unless otherwise noted, which roughly corresponds to the concentration of a genomically-integrated CRISPRi system in actively growing *E. coli* cells.

Parameter	Typical Value	Units
gRNA Production (α_g)	5	min^{-1}
dCas Production (α_C)	1	min^{-1}
gRNA/dCas Binding ($k_{g:c}^f$)	$\frac{\ln 2}{375}$ ($\approx 1.8e - 3$)	$\text{nM}^{-1}\text{sec}^{-1}$
gRNA/dCas Unbinding ($k_{g:c}^r$)	0	sec^{-1}
gRNA:dCas/Promoter Binding ($k_{c:p}^f$)	$\frac{\ln 2}{60}$ ($\approx 1.2e - 2$)	$\text{nM}^{-1}\text{sec}^{-1}$
gRNA:dCas/Promoter Unbinding ($k_{c:p}^r$)	0	sec^{-1}
gRNA Degradation	$\frac{\log 2}{100}$ ($\approx 6.9e - 3$)	sec^{-1}
Dilution	$\frac{\log 2}{30}$ ($\approx 2.3e - 2$)	min^{-1}

Table 3.1: Typical parameter values used for simulating CRISPRi circuits. Parameters are estimated from the literature, except where noted in the text

It is worth mentioning that there is still a great deal of uncertainty around the kinetics of dCas binding. The rates given above for binding and unbinding of gRNAs to dCas

were taken from measurements of dCas/gRNA association rates in vitro (Mekler et al., 2016). It is unclear how closely this estimate follows in vivo kinetics. For example, the same authors show that the addition of total human lung RNA to an in vitro dCas:gRNA assembly reaction slowed dCas:gRNA binding by at least an order of magnitude. Therefore, the estimate in Table 3.1 may be an optimistic one.

Rates of association between gRNA-loaded dCas and its DNA targets have been more widely studied, but there the literature is still conflicted on their actual values. For example, (Gong et al., 2017) report an unbinding rate of dCas from DNA of about $1/(6.5 \text{ min})$ in a radiolabeled pulse chase assay, but that is incompatible with the observation of (Jones et al., 2017) that dCas dissociation in vivo is driven by cell division, or with the real-time, single-molecule measurements of (Boyle et al., 2017), who could not observe sufficient unbinding events over several hours to estimate an unbinding rate for matched gRNAs (setting an upper bound on unbinding time on the order of hours). The parameters for dCas:DNA interactions chosen in Table 3.1 use binding rates from Mekler et al., and reflect the canonical understanding in the field that, for all practical purposes, dCas does not unbind from DNA.

3.3 Modeling Results

The full CRISPRi model predicts that a variety of dynamic circuits can be constructed from CRISPRi, including a repressilator, a toggle switch, a pulse generating type I incoherent feed forward loop (IFFL), and multiple IFFLs independently driven by a 5-node oscillator (Figure 3.2). However, these circuits do not operate well under all possible (or even all “reasonable”) parameter values.

To better understand the parameter requirements of CRISPRi circuits, and the kinds of engineering we might perform to improve CRISPRi circuit robustness, we will start by using a simple approximation of the CRISPRi model before moving to simulations using a more complete model under varying parameters.

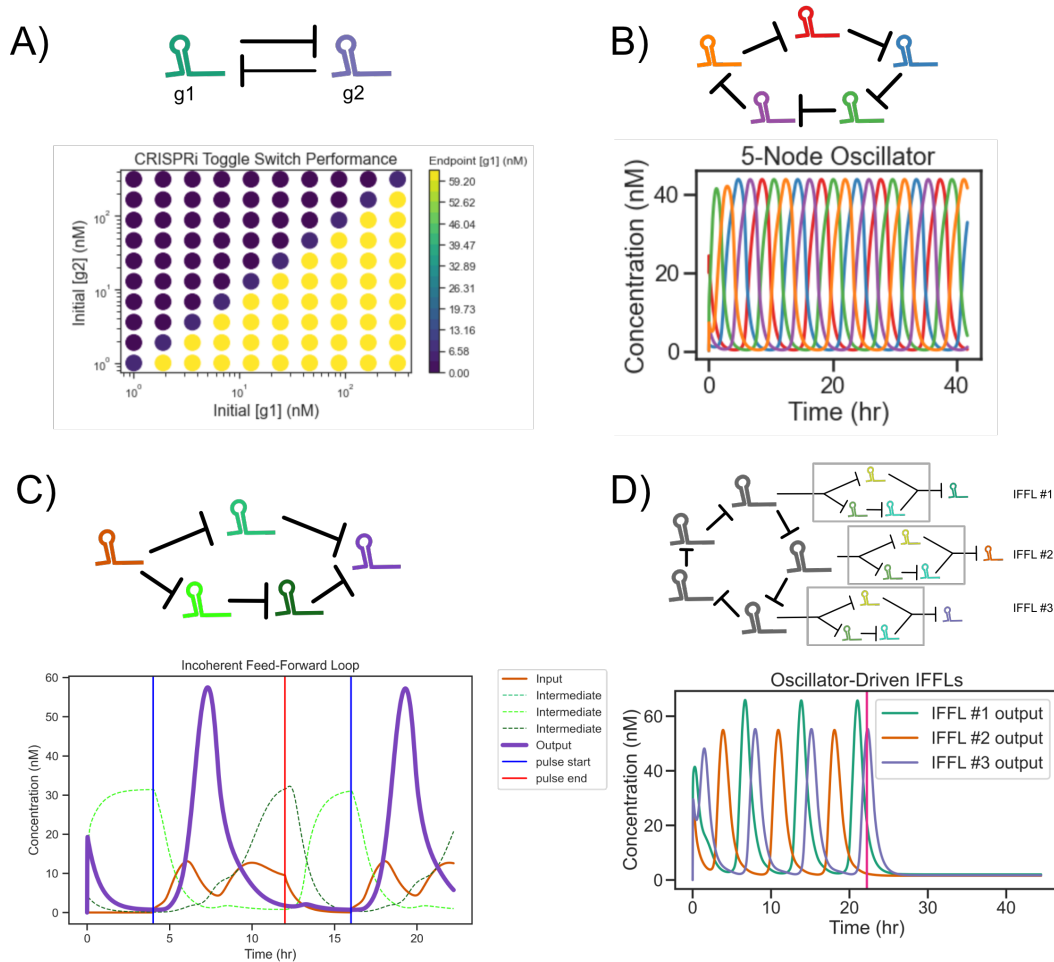


Figure 3.2: Four examples of dynamic circuits made from CRISPRi components, simulated with the full CRISPRi model. Nodes in circuit diagrams represent gRNA expression units; blunted arrows represent dCas-mediated repression. (a) Steady states of a toggle switch for a variety of initial conditions of gRNA concentration. (b) A 5-node oscillator. (c) A type-I IFFL (pulse generator). The purple trace tracks the IFFL output. Vertical blue and red lines mark activation and return to baseline of the input gRNA promoter, respectively. (d) Outputs of three IFFLs driven by a 5-node oscillator. When the node of the oscillator driving the IFFLs is removed (vertical red line), pulses cease. Note that not all nodes of the oscillator have corresponding visible IFFL outputs, and that the peak heights of the three IFFLs are not symmetric.

An approximation

Can we understand CRISPRi dynamics in rational, analytical terms? Should we expect an oscillator made from CRISPRi components to actually oscillate? A toggle switch? An IFFL? According to traditional genetic circuit analysis, the toggle switch (Gardner, Cantor, and J. J. Collins, 2000) and repressilators (Elowitz and Leibler, 2000) rely on cooperative binding. There is no obvious “cooperative” mechanism in the CRISPRi model, so we might wonder whether we should expect these circuits to function at all.

Unfortunately, the full CRISPRi model outlined in Section 2.2 is not particularly amenable to analysis—even the steady state binding between a single gRNA, dCas, and the gRNA’s target is barely analytically tractable without making an unrealistic quasi-steady state assumption (finding it requires the roots of a rather messy fourth-order polynomial). To attempt to make some headway, we split the model into those parts making up an “idealized,” easy-to-analyze CRISPRi process (informally, “first-order” considerations, though this should not be taken to imply linearity) and kinetic considerations that make CRISPRi difficult to analyze (“second-order” considerations).

We propose the following assumptions for a first-order CRISPRi model: dCas is always present in abundance relative to both DNA targets and gRNAs; binding between gRNAs, dCas, and DNA is instantaneous; and binding of dCas to DNA targets is irreversible. Under these assumptions, the binding of gRNA to dCas to target DNA reduces to a simple “linear” model—with increasing concentrations of gRNA, dCas binds 1:1 with DNA until the DNA is completely saturated. This simplification obviously neglects some important features of CRISPRi (binding kinetics and loading effects on dCas, to name two), but it can still provide insights into how CRISPRi circuits work (or don’t).

Consider a CRISPRi toggle switch consisting of two gRNAs repressing each other. The first-order model of the CRISPRi toggle switch can be modeled with just two differential equations:

$$\begin{aligned}\frac{dg_1}{dt} &= \alpha \max(0, P_1 - g_2) + \alpha_0 \min(P_1, g_2) - \gamma g_1, \\ \frac{dg_2}{dt} &= \alpha \max(0, P_2 - g_1) + \alpha_0 \min(P_2, g_1) - \gamma g_2,\end{aligned}$$

Here, g_1 and g_2 are concentrations of two mutually-repressing gRNAs, P_1 and P_2 are total concentrations of promoters for those guides, α is the production rate of gRNA from an unbound promoter, α_0 is the production rate of gRNA from a bound promoter (leak), and γ is the division rate of the cell (dilution).

Under what conditions does this system admit two stable steady states? To answer this, we should consider the intermediate steady state of the system, far from the

bounds set by 0, P_1 , and P_2 . In general, toggle-switch-like circuits undergo a supercritical pitchfork bifurcation at this point. When it is stable, the system admits only one state (Figure 3.3A), but when it is unstable, the system will have two steady states (the “toggleable” steady states) (Figure 3.3B). In particular, the middle steady state will be unstable (and the toggle switch will correctly “toggle”) if and only if that system has a single non-trivial steady state that is unstable. This corresponds to the case where at least one of the eigenvalues of the Jacobian of the system has positive real part. To find when this is true, we note that far from any saturating bounds (where we are likely to find the central steady state), the system reduces to

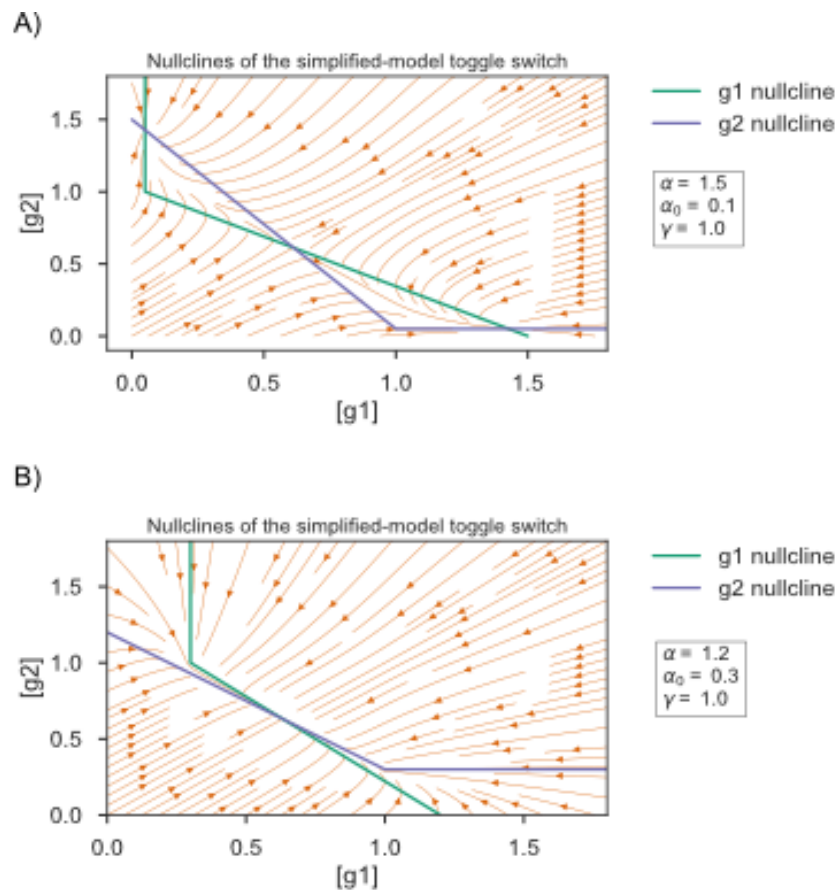


Figure 3.3: Flow fields for the first-order approximation model of a CRISPRi toggle switch. Depending on the parameters chosen, the toggle could (a) admit two stable steady states separated by an unstable steady state or (b) admit a single stable steady state.

$$\begin{aligned}\frac{dg_1}{dt} &= \alpha P_1 g_2 (\alpha - \alpha_0) - \gamma g_1, \\ \frac{dg_2}{dt} &= \alpha P_2 g_1 (\alpha - \alpha_0) - \gamma g_2,\end{aligned}$$

whose Jacobian has eigenvalues $-(\alpha - \alpha_0) - \gamma$ and $(\alpha - \alpha_0) - \gamma$. The first eigenvalue always has negative real part. The first eigenvalue has positive real part (and the system “toggles”) when $\alpha - \alpha_0 > \gamma$. In short, a toggle should function as long as the difference between production rates of bound and unbound promoters is sufficiently large relative to dilution.

We can apply a similar analysis to a three-node CRISPRi repressilator, which is an oscillator consisting of an odd number of guide RNAs in a circular circuit topology, each gRNA repressing the next in the cycle (similar to Figure 3.2B). Bounded dynamical systems with repressilator-like architecture typically have a single non-trivial steady state. As with the toggle, the desired behavior (oscillations, in this case) can occur only when that central steady state is unstable. In principle, an unstable central steady state is not sufficient to guarantee oscillations; in practice, molecular species concentrations are bounded by dilution, and there are no other possible stable steady states to the repressilator system, which leaves little room for non-oscillatory (or chaotic) behavior.

The Jacobian for a three-node CRISPRi repressilator has eigenvalues $\frac{1}{2} \left(\pm \sqrt{-3(\alpha - \alpha_0)^2 + (\alpha - \alpha_0) - 2\gamma} \right)$ and $-(\alpha - \alpha_0) - \gamma$. The last eigenvalue always has negative real part. The first pair of eigenvalues each have positive real part (and the system oscillates) when $\frac{\alpha - \alpha_0}{2} > \gamma$. As with the toggle switch, the difference between production rates of bound and unbound promoters must be sufficiently great for the CRISPRi repressilator to oscillate.

We will soon see that this simplified model is a poor predictor of exactly what behavior a CRISPRi circuit with particular parameters or will not exhibit. This is beside the point; what we learn from the simplified CRISPRi model is that cooperativity is *not* necessary for either a toggle switch or a repressilator, despite classical understanding in the literature (Gardner, Cantor, and J. J. Collins, 2000; El-Samad, Vecchio, and Khammash, 2005). Cooperativity, it seems, is necessary only when genes are expected to bind in a Hill-like fashion. Perfectly linear binding with sharp saturation, as we should expect in CRISPRi, is another perfectly viable path to useful instability.

Note that the result that a CRISPRi oscillator does in fact oscillate contradicts the modeling results by (Santos-Moreno, Tasiudi, et al., 2020). In that work, the authors model a CRISPRi toggle as a slightly more tractable proxy for the oscillator. The authors analyze this model using BioSWITCH, a toolbox for limit point detection and bistability analysis across the space of all “reasonably” bounded parameters.

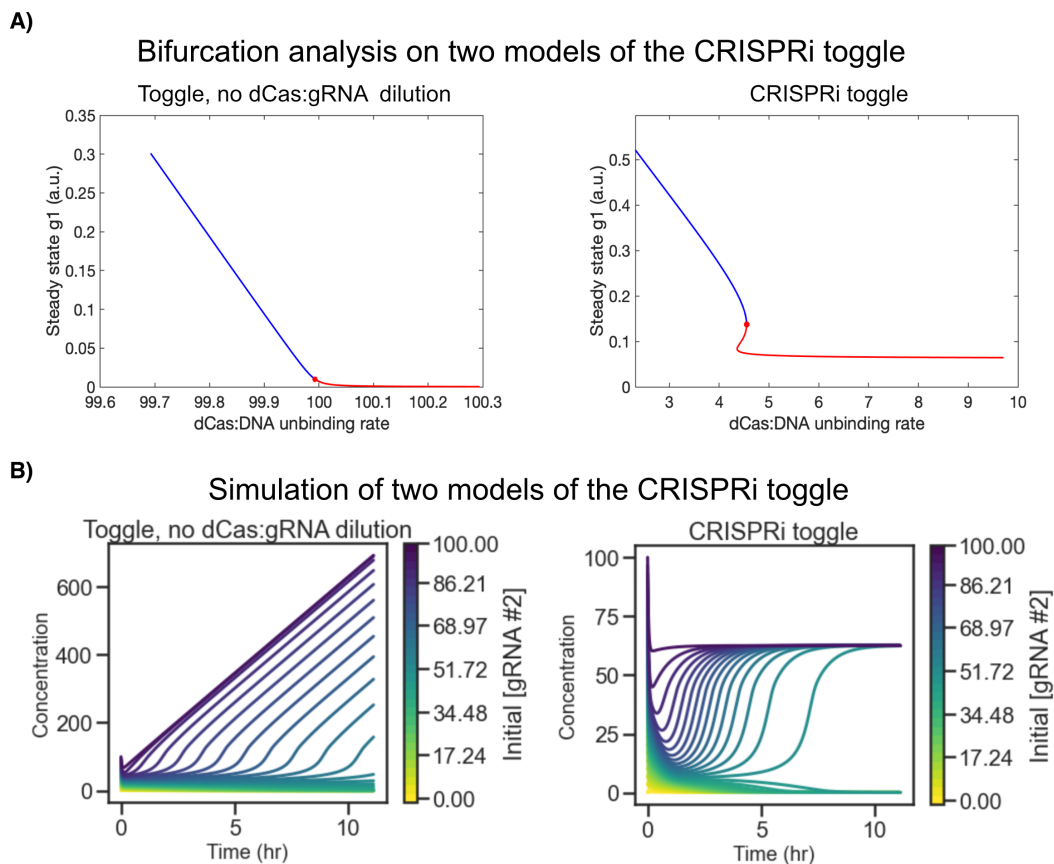


Figure 3.4: Steady-state analysis with the BioSWITCH toolbox shows that bistability is possible with dCas:gRNA complex dilution (right), but not without it (left). Plots show possible steady-state values of gRNA #1 as a function of one parameter, with the others held fixed at values computed to optimally detect bifurcation points.

The authors show that a simple model similar to our full model fails to predict the steady-state instability required for bistability in the toggle (Figure 3.4, left). The authors additionally show that bistability can be recovered by adding off-target and cross-target binding.

This analysis contradicts ours in two ways. Firstly, our model predicts that a CRISPR toggle switch should function even without any off-target or cross binding. Secondly, our model predicts that off-target binding should *decrease* the robustness of a CRISPRator, not increase it (Figure 3.5A).

One difference between our model and that of Santos-Moreno and Taisiudi is that our model includes degradation of gRNA-bound dCas complexes. Adding these degradation reactions to their model is sufficient to allow steady-state instability according to BioSWITCH (Figure 3.4A). Our own model shows that with gRNA:dCas dilution removed, the toggle switch's “on” steady state becomes unstable, causing the high-state gRNA to increase without bound (Figure 3.4B).

Now that we have some theoretical justification for believing that a CRISPRi toggle

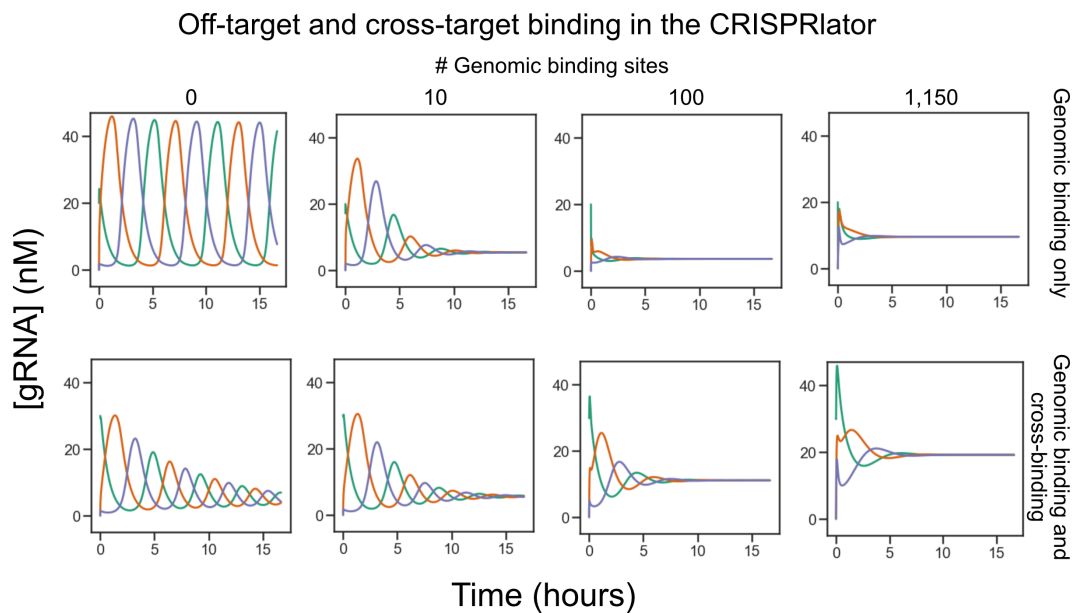


Figure 3.5: Addition of off-target binding destroys oscillations in the 3-node CRISPRlator, contra Santos-Moreno, Taisiudi, et al., 2020. In the top row, additional non-target binding sites have been added. In the bottom row, additional non-target binding sites have been added *and* guides can bind at a low rate to mismatched target sites.

switch or repressilator should be possible to build, we will explore what conditions allow those circuits to function under the full CRISPRi model.

Repressilators can be made with CRISPRi, but they can display strong initial condition dependence

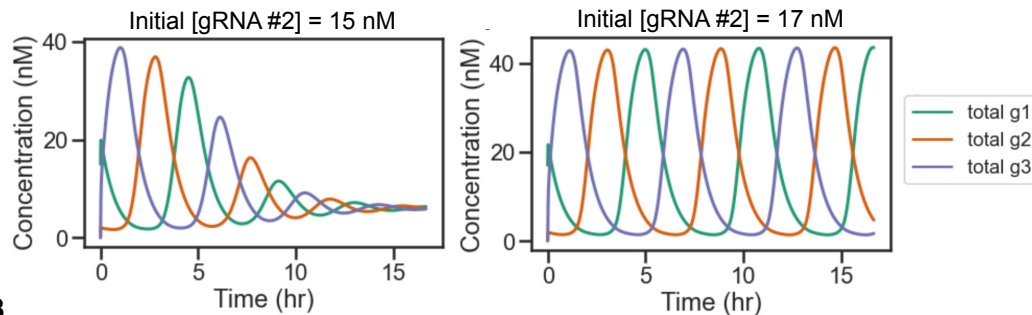
Consider, again, the 3-node CRISPRi repressilator, or CRISPRlator. Our model predicts it to be quite slow, with a period of about seven hours (about three times as long as the original protein-based repressilator (Elowitz and Leibler, 2000)). This is because where the time scale of the repressilator is limited by active protein degradation, the timescale of the CRISPRlator is set by dilution.

Interestingly, even under parameterizations that allow oscillations, the CRISPRlator does not oscillate for all initial conditions (Figure 3.6). It is possible for the 3-node CRISPRlator to possess both a stable limit cycle and an unstable limit cycle inside the stable limit cycle which screens off trajectories with insufficient differences in concentrations of different gRNAs. These latter trajectories, which fall inside the unstable limit cycle, spiral to a stable steady state.

Note that this behavior differs from the behavior of the classic repressilator, which (in its reduced three-species form) can only have up to a single, stable limit cycle (V. P. Golubiatnikov and Ivanov, 2018; Likhoshvai, Vladimir P. Golubiatnikov, and Klebodarova, 2020). Also note that the existence of multiple limit cycles in the CRISPRlator implies that linearization characterization of the steady state of

a CRISPRiator is insufficient to determine whether the CRISPRiator will actually oscillate. If the CRISPRiator's steady state is unstable, then it will oscillate; but if the steady state is *stable*, then it could either have a pair of stable/unstable limit cycles (in which case it can oscillate), or no limit cycles (in which case it will not). Accordingly, when we computationally screen for oscillations in various CRISPRiators, as in Figure 3.7, we do so using numeric simulation and heuristic oscillation detection rather than with stability analysis.

A) Two identical CRISPRiators with slight differences in initial condition



B, Multiple limit cycles in a 3-node CRISPRiator

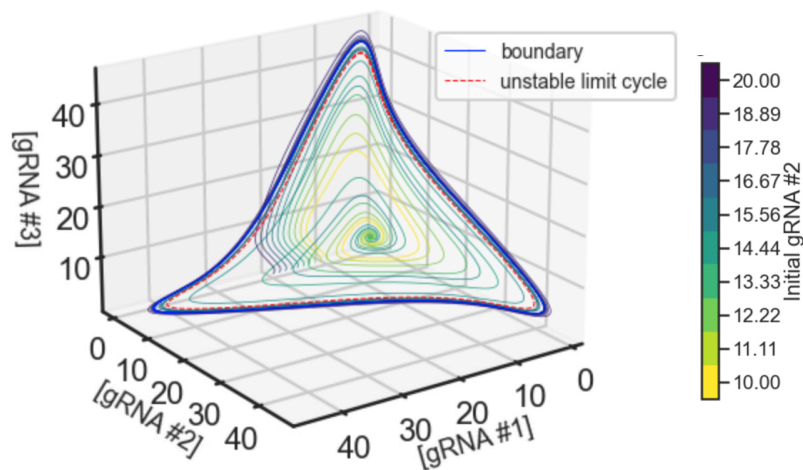


Figure 3.6: **(a)**The same CRISPRiator (with the same rate parameters) oscillates for some initial conditions but not for others. In both cases, dCas begins at 43 nM and all other non-DNA species other than gRNA #2 start at 0 nM. **(b)** Trajectories for a variety of initial conditions reveal multiple nested limit cycles in the 3-node CRISPRiator under some parameterizations. In this example, an unstable limit cycle (dotted red loop) screens trajectories with too little total gRNA from a stable limit cycle (solid blue loop).

The CRISPRiator is fragile

Unfortunately, the CRISPRi repressilator is fragile, and appears to sit close to a bifurcation in parameter space. It is, for example, fairly sensitive to dCas production rate, and ceases to oscillate with less than about 10% or more than about 150% of

the default dCas production rate in Table 3.1. This indicates that dCas production levels may have to be fine-tuned specifically for any particular CRISPRi circuit. Luckily, this is a relatively easy parameter to tune in a real cell.

The repressilator is also not generally robust against transcriptional leak. The first-order model predicts that an increase in leak should stabilize the system towards a steady state, eventually driving it to equilibrium with no oscillations. This is reflected in the full model, and not only for the CRISPRlator—addition of as little as 1% leak destroys oscillations in all but one of the simulations shown in Figure 3.2 (the exception is the CRISPRlator-driven triple IFFL of Figure 3.2D, which breaks between 2 and 4% leak).

That a CRISPRlator has, in fact, been built successfully suggests that CRISPRi might, in fact, have extremely low leak in the true sense of allowing transcription while the repressor is bound (Santos-Moreno, Taisiudi, et al., 2020; Kuo et al., 2020). Repression with dCas has been reported in cell-free extract with fold-repression between 7 and 100, and in vivo with similar repression strengths, which puts the best CRISPRi repression in a leak range that should not be likely to allow a repressilator to work. Our model of the CRISPRlator suggests that the “leak” observed in these experiments may be a function of *slow binding*, not failure of a bound repressor—that is, observed “leak” occurs because a promoter is not bound some of the time, not because a promoter is bound to dCas and produces transcripts anyway.

Active degradation can offset leak-based circuit fragility

There are a few different knobs we can turn to make the CRISPRi repressilator more robust to transcriptional leakiness. We can decrease the rate of production of either dCas or gRNAs; we can speed the binding between dCas:gRNA complexes and DNA (in contrast, speeding binding between dCas and gRNAs appears to have little effect); we can add active degradation of dCas (also increasing the *speed* of the oscillator considerably); and we can grow the repressilator from three nodes to five nodes.

As an example, let us consider the interaction between dCas degradation rate and leak rate. Figure 3.7 shows the performance of simulations of several circuits as a function of the degradation rate and leak rate parameters, with other parameters as shown in Table 3.1. We can think of these charts as a sort of two-dimensional “specification sheet” for dCas to allow different circuits to function properly. For any particular leak rate, the circuit will either oscillate (Figure 3.7A, B, D, E, and F) or toggle (Figure 3.7C) only when dCas is degraded at a rate within a proper range. Too much dCas degradation will destroy all examined circuits, and a minimum amount of degradation is required for some. Notably, the “correct” dCas degradation specification is different for different circuits (compare Figures 3.7A, 3.7B, and 3.7C), dCas expression rates (Figures 3.7D and E), and background activity (different numbers of oscillators in Figure 3.7F). Interestingly, the toggle switch has similar parameter requirements on these two axes, suggesting that there

may be some requirements shared by some interesting class of dynamic CRISPRi circuits.

We can produce a similar “specification sheet” for dCas degradation and leak for a 5-node CRISPRi repressilator, as shown in Figure 3.7B. The 5-node repressilator is more robust than the 3-node oscillator. Indeed, the 5-node repressilator can operate with as much as 10% leak or as little as no dCas degradation at all. In the case of CRISPRi repressilators, bigger is not only better, but potentially easier.

On the other hand, the parameter requirements of the toggle switch appear to be quite similar to those of the 3-node repressilator (Figure 3.7C). Admittedly, the toggle switch and 3-node repressilator have very similar architecture, but the fact that both circuits require similar degradation rates and minimum promoter leak suggests that the regime of functional repressilators may have not-yet-understood underlying properties that are broadly useful for constructing CRISPRi circuits.

There are more than two tunable knobs in the CRISPRi system. One that we have already seen to be important is the *production* rate of dCas. Figure 3.7D shows how the target parameter set changes with different levels of dCas production. The good news is that with low enough dCas expression there is no need for dCas degradation (though with dCas steady state levels that low, stochastic fluctuations become a more serious problem). The bad news here is that at least one *engineerable* but not *readily tunable* parameter of CRISPRi (namely, dCas degradation rate) has acceptable value ranges that don't overlap for some choices of dCas production rate. This should not be too much of a problem for making a single repressilator, but it does complicate the design and integration of multiple CRISPRi circuits in the same cell. For example, Figure 3.7E shows the expected effect of expressing two identical CRISPRi repressilators in parallel with no directly cross-interacting nodes. The increased load on dCas drops the effective steady-state concentration of dCas as perceived by each individual oscillator, which has a similar effect as dropping dCas production rate. Namely, this shifts the required rate of dCas degradation. A repressilator that works on its own can be expected to fail when a second repressilator is added, unless dCas's degradation rate is exquisitely well-tuned. More generally, it seems likely that different circuits may require dCas variants with different degradation rates.

Reciprocally, we could tune degradation rate to compensate for changes in other parameters. The CRISPRiator requires dCas to be produced at a tuned rate—too much or too little dCas production will destroy the circuit's function. Changing the rate at which dCas is degraded *also* changes those dCas production rate requirements, potentially allowing degradation to compensate for any lack of control over dCas concentration (Figure 3.7D).

Finally, we can consider the robustness of the CRISPRiator to differences in the strengths of the promoters driving gRNA production. In general, repressilators require nodes with roughly similar repression strengths. The more sensitive the CRISPRiator to gRNA promoter strengths, the more difficult a CRISPRiator will be to engineer. Fortunately, as shown in Figure 3.8, the 5-node CRISPRiator is robust to (most) changes of at least 10-fold in two adjacent gRNA.

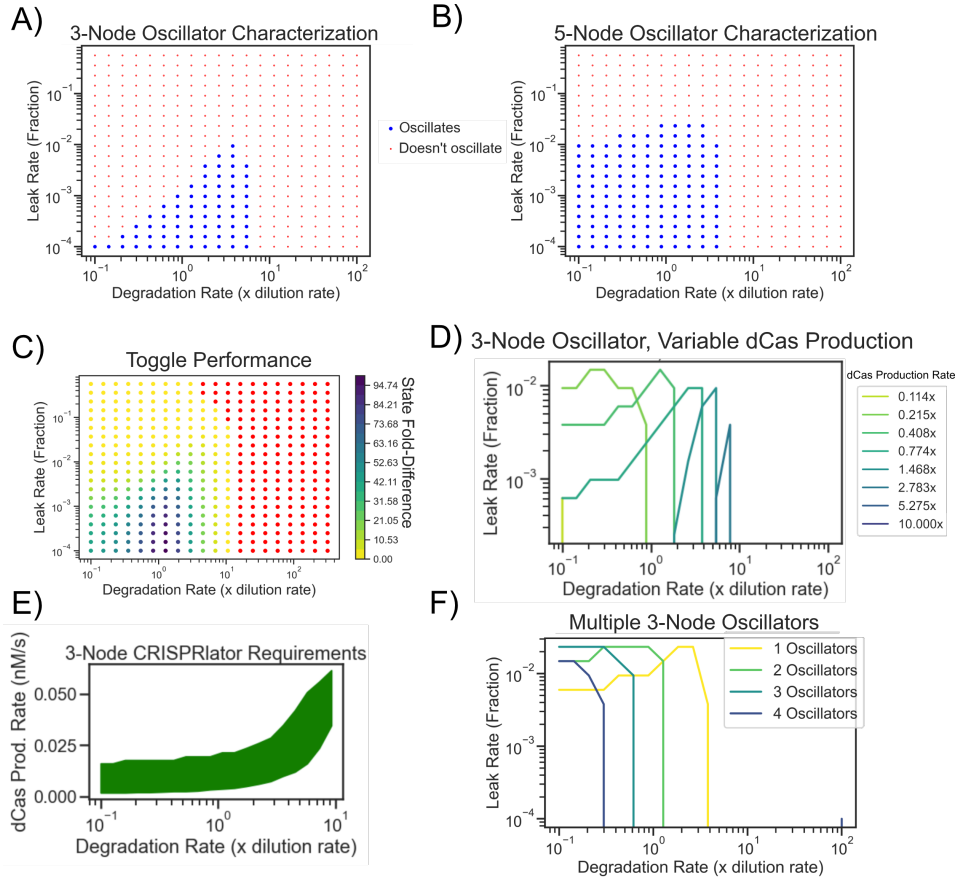


Figure 3.7: Acceptable degradation rates vary by circuit architecture and parameterizations. Leak is given in units of the rate of gRNA production from an unbound promoter. Degradation rate is in units of cell division rate. Colors at each point in parameter space indicate whether the circuit oscillates (blue) or not (red) for those given parameters. Results are given for **a)** a 3-node CRISPRi, **b)** a 5-node oscillator, and **c)** a CRISPRi toggle switch. In **c)**, Colors indicate the separation distance between the two gRNAs at steady state; red indicates no detectable bistability. **d)** Changes in degradation requirements for a 3-node CRISPRi with different levels of dCas expression. Areas under each curve represent parameters for which the circuit oscillates. Production rates are given in units of min^{-1} (the “default” dCas speed used in (textbfa)). **e)** Changes in dCas expression requirements for the 3-node CRISPRi with different rates of dCas9 degradation. The shaded region represents production rates that admit oscillations. Oscillations could not be recovered with any higher degradation rate. Transcriptional leak is set to 0. **f)** Parameters for which different numbers of independent 3-node CRISPRi oscillate while operating in the same cell. Larger number of oscillators become steadily less tolerant of both degradation and leak.

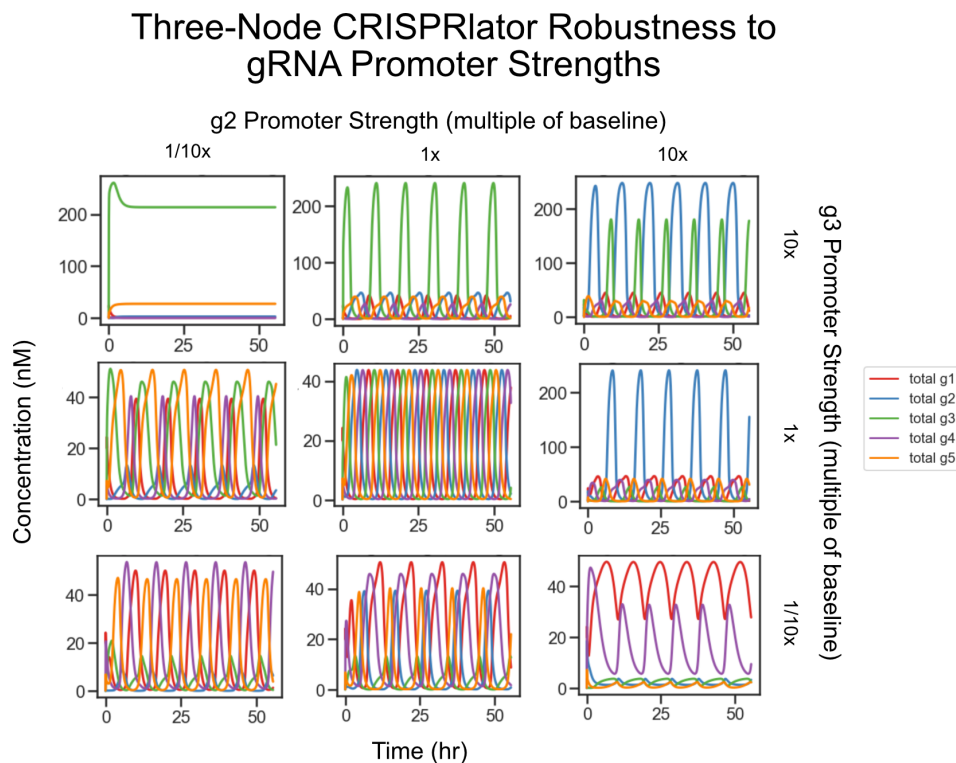


Figure 3.8: The 5-node CRISPRiator functions under many possible changes to the strengths of two adjacent gRNA promoters. Other parameters are as in Figure 3.2b. Each panel shows simulation results with one combination of promoter strengths.

3.4 CRISPR Activators Scale Better Than CRISPR Inhibitors

A crucial limit on the scalability of CRISPRi circuits is bottlenecking of dCas9. Any expression of one gRNA in a network of CRISPR transcription factors sequesters shared dCas9 away from other gRNAs, decreasing their effectiveness as outlined in detail in (Chen, Qian, and Del Vecchio, 2019). The dCas9 bottleneck can be loosened by producing more dCas9, but this strategy is limited by the toxicity of dCas9 when expressed at high concentration, especially in prokaryotes (Cui et al., 2018; Cho et al., 2018).

Zhang and Voigt (2018) quantify this bottlenecking effect in *E. coli*. They show that under physiological, circuit-like conditions, dCas9 bottlenecking causes CRISPRi target repression to drop off as roughly $\frac{1}{N}$, where N is the number of competing gRNAs expressed. Furthermore, they show that *E. coli* expressing near-maximal sustainable levels of dCas9 repressor cannot support more than about ~ 7 simultaneous gRNAs at a 10-fold level of repression (or about ~ 15 simultaneous gRNAs using a low-toxicity dCas9 variant, dCas9*-PhIF).

Prior to 2018, CRISPR transcription factors in bacteria were largely limited to repressors, as existing CRISPR activators typically exhibited ≤ 10 -fold activation and only functioned in an unusual *RpoZ*-knockout strain; accordingly, the Zhang and Voigt analysis of resource bottlenecking reasonably considered only CRISPRi.

There are now more effective CRISPRa activators made using other activators that do not require *RpoZ* knockout, which makes CRISPRa a feasible alternative to CRISPRi for synthetic gene circuit construction (Dong et al., 2018; Ho et al., 2020).

In this section, we use a simple model of gRNA competition for dCas9 to show that CRISPR activators are substantially less vulnerable than CRISPR repressors to dCas9 bottlenecking. Our model anticipates that CRISPRa should be able to support many times more simultaneous gRNAs than CRISPRi under most conditions, although CRISPRi may be more effective for very small networks under ideal conditions.

Why activators are less sensitive to bottlenecking than repressors

CRISPR activators should scale better than CRISPR repressors because, in general, activators are not affected as badly as repressors by removal of a small amount of regulator when they are already saturating.

Consider a repressor and an activator of equal strength and at high enough concentration that every target promoter is bound. What happens if the concentration of each regulator drops enough that one of the targets becomes unbound? How much is each system affected? We could equivalently consider the fraction-of-time-bound for a regulator with a single-copy target, but for conceptual simplicity, we will consider a system with a “large number” (say > 5) of targets that are each either bound or not.

When one activator drops off a target, total expression falls by a little less than a fraction $\frac{1}{N}$ of maximum expression, where N is the number of target promoters. The fold change in expression caused in this decrease in activators is therefore roughly $\frac{1}{1-\frac{1}{N}} = \frac{N}{N-1} = 1 + \frac{1}{N-1}$. If N is reasonably large, this fold change will be quite small (consider $N = 10$).

When all target promoters are bound by repressors, the total expression of the bound promoters is $\frac{1}{R}$, where R is the fold-repression of a single repressor binding to a single promoter. When a single repressor is removed, the change (increase, this time) in expression is, again, roughly $\frac{1}{N}$, but this change occurs against a background of “leak” experienced when all repressors are bound (which is hopefully a small value) rather than maximum possible promoter expression (which is hopefully a larger value). The fold change experienced due to this removal is roughly $\frac{\frac{1}{N} + \frac{1}{F}}{\frac{1}{F}} = 1 + \frac{F}{N}$, where F is the fold-change in expression between bound and unbound promoter (i.e., $F = 10$ for a repressor that reduces its target to 1/10th its constitutive strength when bound). This fold change is only small if the number of targets is large compared to the fold-repression of the repressor, and for most reasonably capable repressors in most reasonably-sized cells it will be at least 2-fold.

Viewed another way, the changes in expression experienced when a small amount of activator or repressor are removed are of roughly the same *absolute* magnitude, but the relative impacts of those effect are quite different. For an activator, the change should be compared against maximum expression (which should be large), whereas



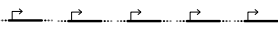
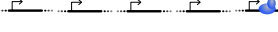


Occupancy state:	Expression when regulator is a:	
	Repressor	Activator
<i>One Promoter</i>		
	100%	10%
	10%	100%
Expression Change	10x	10x
<i>Low Occupancy</i>		
	100%	10%
	82%	28%
Expression Change	1.2x	2.8x
<i>High Occupancy</i>		
	28%	100%
	10%	82%
Expression Change	2.8x	1.2x

Figure 3.9: Activators and repressors respond differently to changes in regulator concentration in highly saturated and highly unsaturated conditions. Both the repressor and the activator change expression 10-fold (top table). At low saturation of target promoter by the regulators, the activated promoter is more sensitive to changes in regulator concentration than the regulated promoter (middle table). Conversely, at high target saturation, the activated promoter is more robust to changes in regulator concentration (bottom table).

for a repressor, the change should be compared against promoter leak (which should be small). Figure 3.9 shows this difference for a concrete example of a 10-fold activator and a 10-fold repressor targeting a 5-copy promoter.

Note that the asymmetry between sensitivities of activators and repressors reverses at low concentrations of regulator. There, activators are more sensitive to regulator changes and repressors are more robust. Compare Figure 3.9, middle and bottom rows. Activators trade off high sensitivity to concentration changes at low saturation for robustness to concentration changes at high saturation, while repressors make the opposite tradeoff.

Simulations show that CRISPRa scales better than CRISPRi

Using a steady-state solution for a simple ODE model of CRISPR, we numerically investigated the effects of gRNA competition on minimal CRISPRi and CRISPRa systems consisting of a repressing or activating dCas9:gRNA complex targeting a reporter gene. See Section 3.2 for a description of the model and details on how we calculate fold change.

We calculate fold change for three different concentrations of target promoter roughly representing a genomically-integrated reporter, a reporter on a low-copy plasmid, and a reporter on a high-copy plasmid (Figure 3.10A, B, and C, respectively). We use parameters estimated for this model from in vivo data by Zhang and Voigt (2018). We vary the concentration of dCas9 from 100 nM up to 530 nM, which is roughly the maximum concentration of dCas9 the *E. coli* used by Zhang and Voigt can support before suffering significant growth defects.

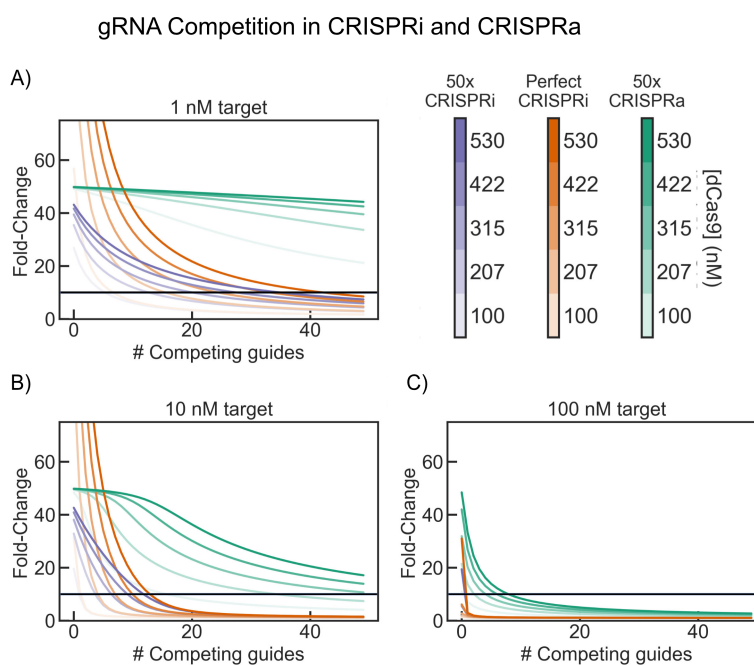


Figure 3.10: Simulated fold change of activation with CRISPRa (green) or repression with CRISPRi (perfect repression, orange; or 50x repression, purple) with various dCas9 concentrations and either (a) 1 nM, (b) 10 nM, or (c) 100 nM of target promoter. Parameters taken from (Zhang and Christopher A. Voigt, 2018). The horizontal line in each plot marks the 10x fold change.

The same model that (correctly) predicts that the scalability of CRISPRi is severely hampered by inter-gRNA competition also predicts that CRISPRa should be far more robust against inter-gRNA competition.

We also predict that CRISPRi may produce higher-fold changes than CRISPRa at low numbers of competing guide RNAs, especially when dCas9 is abundant with respect to the target promoter. Note that this prediction is a direct consequence of the assumption that dCas9 is a perfect repressor, but a finite activator. This means that the effectiveness of CRISPRi repression is unbounded above—as binding becomes more efficient, the fold change of repression approaches infinity—while CRISPRa activation effectiveness is bounded above at 50x. If we relax this assumption so that CRISPRi is equally as “effective” as CRISPRa (i.e., it represses 50x when

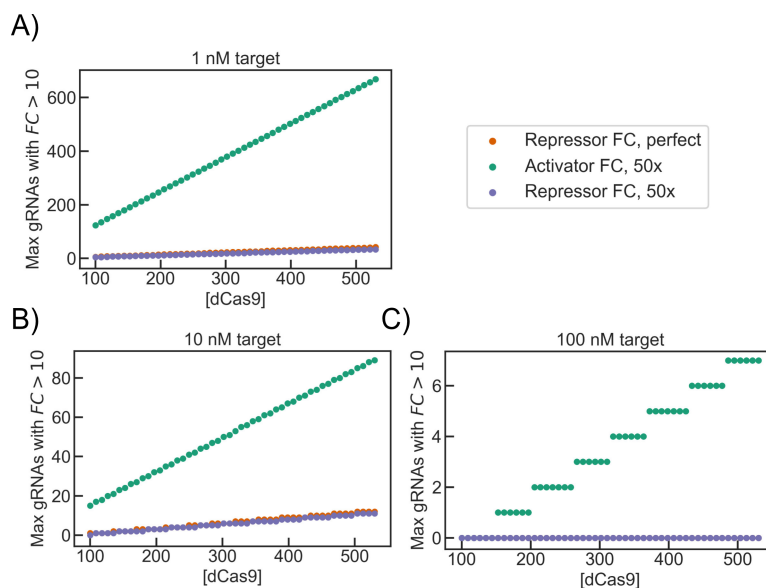


Figure 3.11: Predicted maximum number of competing guide RNAs before competition reduces fold change to below 10x, for various dCas9 concentrations and (a) 1 nM, (b) 10 nM, or (c) 100 nM of target promoter. Parameters taken from (Zhang and Christopher A. Voigt, 2018). “0 Max gRNA” means that a single gRNA competitor is sufficient to drive fold change below 10x.

fully bound), then it fails to exceed (or even match) the overall fold-repression of CRISPRa at *any* number of competing gRNAs (Figure 3.10, purple curves).

These results are not unduly sensitive to the particular parameters estimated in Zhang and Christopher A. Voigt, 2018. We computed maximum “acceptable” competing gRNA number (the largest number of competing gRNAs that still allows 10-fold regulation of the target) for 1,000 randomly sampled parameters (see Figure 3.12 for results from 100 representative parameter sets; see figure legend for details on parameter sampling). Although there was a significant degree of variation in CRISPRa scalability across simulations, only rarely did we observe CRISPRa with worse scaling than any simulated CRISPRi system (Figure 3.12B).

The model

We adapt the modeling framework derived in (Chen, Qian, and Del Vecchio, 2019) and adapted by Zhang and Voigt in their analysis of gRNA competition in CRISPRi. We recapitulate their analysis and extend it to CRISPRa.

Following the analysis in (Zhang and Christopher A. Voigt, 2018) (originally formulated in (Chen, Qian, and Del Vecchio, 2019)), we will consider a model of dCas9 repression in which a gRNA g_1 is in competition with some number N of functionally identical, non-targeting gRNAs for a fixed amount of core dCas9. We wish to calculate the fold change of repression of the target of g_1 as a function of N .

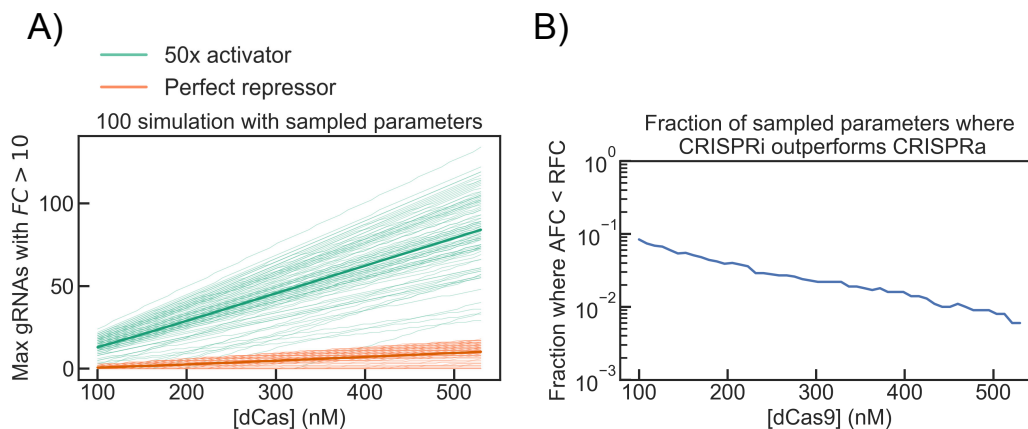


Figure 3.12: **(a)** Predicted maximum number of competing guide RNAs that still allow 10-fold regulation for 100 of the 1,000 tested CRISPRi/a systems with randomly sampled parameters. Thin lines represent single sampled parameter sets; bold lines represent averages across all sampled parameter sets. Target copy number was held fixed at 10. Transcription speed for competing gRNAs was held to a constant multiple of that of the primary gRNA (changes in one transcription rate relative to another are equivalent to a compression or expansion of the gRNA concentration axis). Fold-activation of the CRISPRa activator was sampled from a normal distribution with mean 50x and a 20% standard deviation, reflecting our high confidence in our estimates of that parameter from (Dong et al., 2018). All other parameters were log-normally distributed around their best-estimate values, as determined in (Zhang and Christopher A. Voigt, 2018), with a \log_{10} standard deviation of 0.5, very roughly representing a half-order-of-magnitude “50% confidence range.” **(b)** The fraction of sampled parameter sets (out of 1,000) in which CRISPRa was predicted to perform better than *any* sampled CRISPRi, at each possible dCas9 concentration (CRISPRi was never observed to perform better than CRISPRa with the same parameters).

We model dCas9 as a Shea-Ackers repressor of cooperativity 1 and a single binding site per promoter (Shea and Ackers, 1985). We can then write down the average transcription rate of a target promoter as a sum of rates r_f and r_C of transcription from free and dCas9-bound promoter, respectively, weighted by the concentration of promoter in each of those states:

$$r = r_f P + r_C P_C, \quad (3.1)$$

where r is the average transcription rate from the target promoter. If no gRNA is expressed, $P_C = 0$ and Equation 3.1 simplifies to $r = r_f P_{tot}$. We assume that $r_C = 0$ (i.e., repression by bound dCas9 is perfect, allowing no leak), so when the gRNA is active, Equation 3.1 simplifies to $r_f P$, where P will vary with dCas9 concentration, gRNA expression level, number of competing gRNAs, etc. We can then write the fold change of repression (RFC) as

Parameter	Description	Value	Units
r_f	Transcription rate from free (unbound) promoter.	various	s^{-1}
r_C	Transcription rate from dCas-bound promoter	various	s^{-1}
K	Association constant for dCas:gRNA complex binding to promoter.	2.9	$nM^{-1}sec^{-1}$
P_{tot}	Total promoter concentration.	various	nM
α_1	Transcription rate of g1	7.6×10^{-3}	nM/s
α_x	Transcription rate of competing gRNAs.	2.3×10^{-2}	nM/s
β	$\frac{\delta}{K_1}$, δ =(degradation plus dilution) rate for gRNAs.	3.0×10^{-2}	$nM^{-1}s^{-1}$
C_{tot}	Total steady-state dCas pool.	various	nM
N	Number of competing gRNAs (not including g1).	various	unitless

Table 3.2: Parameter notation and values used in this model. Where possible, values were taken from fits to endpoint repression data in (Zhang and Christopher A. Voigt, 2018) (see Figure 3 of that paper).

Variable	Description
P	Concentration of free promoter.
P_C	Concentration of promoter bound to dCas9 repressor or activator.
C_{g1}	Concentration of dCas9:g1 complex.
C	Concentration of free dCas9 regulator.

Table 3.3: Dynamic variables used in our model.

$$RFC = \frac{r_f P_{tot}}{r_f P}. \quad (3.2)$$

Now we can solve RFC at steady-state. We assume total promoter concentration is held constant, so $P_{tot} = P + P_C$. At steady state, there will be flux balance between dCas9 binding to P and dCas9 unbinding from P_C , so $P_C = KC_{g1}P$, where $K = \frac{k_f}{k_r}$ is the association constant of binding between dCas9:gRNA complex and free promoter. RFC then becomes

$$RFC = \frac{r_f P_{tot}}{r_f P} = \frac{r_f(P + KC_{g1}P)}{r_f P} = 1 + KC_{g1}. \quad (3.3)$$

Now we make the informal quasi-steady state assumption that total dCas9 concentration C_{tot} is a constant set upstream by dCas9 production rates and dilution/degradation. Using a conservation law for dCas9 and steady-state flux balance of dCas9 and guide RNAs, the steady-state concentration of dCas9:g1 (C_{g1}) can be calculated as

$$C_{g1} = \frac{\alpha_1 C_{tot}}{\beta + \alpha_1 + N\alpha_x} - P_C. \quad (3.4)$$

To find the steady-state concentration of P_C , we use 1) flux-balance between bound and free promoter and 2) mass conservation of promoter:

$$P_C = KC_{g1}P, \quad (3.5)$$

$$P_{tot} = P_C + P. \quad (3.6)$$

Combining these and rearranging to solve for P_C , we get

$$P_C = \frac{KC_S P_{tot}}{1 + KC_{g1}}, \quad (3.7)$$

where, again, K is the *association* constant for binding between dCas9:gRNA and promoter. Substituting this into (3.4) and rearranging yields a quadratic polynomial in C_S :

$$0 = -KC_{g1}^2 + C_{g1}(KC^* - KP_{tot} - 1) + C^*, \quad (3.8)$$

where

$$C^* = \frac{\alpha_1 C_{tot}}{\beta + \alpha_1 + N\alpha_X}. \quad (3.9)$$

C_{g1} is then well-specified algebraically, and can easily be computed numerically.

A simple model of CRISPRa

We apply a similar analysis to the case of an activating CRISPR system under varying gRNA competition. The most salient changes from the analysis in Section 3.4 are 1) $r_C \neq 0$ (and $r_f \neq 0$), and 2) the fold change we wish to calculate is the reciprocal of that in the CRISPRi case. This gives a fold change of activation (AFC) of

$$AFC = \frac{r_f P + r_C KC_{g1} P}{r_f (P + KC_{g1} P)} = \frac{1 + \frac{r_C}{r_f} KC_{g1}}{1 + KC_{g1}}, \quad (3.10)$$

again with C_{g1} specified by Equations 3.8 and 3.9. We are left with an additional parameter in the activation case, the ratio of activation $\frac{r_C}{r_f} = r_A$ for a single target promoter by a bound activator. This is because we cannot assume zero expression from an un-activated promoter the way we assumed perfect repression of the bound promoter in CRISPRi; in practice, engineered activatable CRISPRa promoters are built from weak, but functional, constitutive core promoters that produce some transcription when unbound. We estimate an activation ratio of 50:1 based on the optimized dCas9-SoxS_{R93A} activation system from (Dong et al., 2018).

Sensitivity to Parameters

We investigated the robustness of our results to errors in parameters by subjecting our model to a local, gradient-based sensitivity analysis, using two summary statistics of CRISPRa performance relative to CRISPRi. We define the “fold change overperformance” of a pair of CRISPRa/CRISPRi systems with a specific set of parameters (including a specific number of competing guides) as the fold change of activation for a CRISPRa system with that set of parameters divided by the fold change of repression for a CRISPRi system with that set of parameters:

$$\text{Fold Change Overperformance}(\text{params}) = \frac{AFC(\text{params})}{RFC(\text{params})}.$$

We also measure the advantage in scalability of CRISPRa over a CRISPRi system with the same parameters with a measure we call “scaling overperformance,” which we define as the maximum number of competing gRNAs that the system can tolerate with fold change caused by g_1 remaining above 10-fold for the CRISPRa system, divided by the same number for the CRISPRi system:

$$\text{Scaling Overperformance}(\text{params}) = \frac{\max(N) \text{ s.t. } AFC > 10}{\max(N) \text{ s.t. } RFC > 10}.$$

For each overperformance measure, we numerically calculate sensitivity of that measure to each parameter as the derivative of overperformance with respect to that parameter at our best-guess parameter set. These raw sensitivity values were normalized against parameter scale (i.e., errors in parameter estimation are assumed to be proportional in scale to the values of those parameters) and overperformance at the best-guess parameter value (so that sensitivity is given as a relative error):

$$\text{sensitivity}(\text{params}) = \frac{\text{params} * \text{raw sensitivity}(\text{params})}{\text{overperformance}(\text{params})}.$$

For $N = 30$ competing gRNAs, $P_{tot} = 10$ target copies, $C_{tot} = 530$ copies of dCas9 (roughly the maximum number an *E. coli* cell can support with negligible growth defect), $r_A = 50$, and all other parameters set as in Table 3.2, we find the sensitivities of fold change overperformance and scaling overperformance listed in Tables 3.4 and 3.5, respectively.

Unsurprisingly, fold change overperformance is most sensitive to r_A , the fold change of activation for promoters bound to a CRISPR activator. Fold change overperformance is also somewhat sensitive to C_{tot} , P_{tot} , α_1 , and α_x . We have already shown how CRISPRi and CRISPRa performances change over realistic values of C_{tot} and P_{tot} .

The parameters α_1 and α_x —the transcriptional speeds of the target gRNA and competing gRNAs, respectively—are effectively different ways of scaling N . Increasing

Parameter	Sensitivity (normalized)	Description
r_A	0.848310	Fold change of activation
C_{tot}	0.716537	Total steady-state dCas pool
α_1	0.709052	Transcription rate of g1
P_{tot}	-0.690079	Target promoter concentration
α_x	-0.679508	Transcription rate of other gRNAs
β	-0.029544	$\frac{\delta}{K_1}$, δ =(degradation plus dilution) rate for gRNAs
K	0.026457	Association constant for dCas:gRNA complex binding to promoter.

Table 3.4: Sensitivity of fold change overperformance to each parameter.

Parameter	Sensitivity (normalized)	Description
r_A	1.079925	Fold change of activation
C_{tot}	-0.335800	Total steady-state dCas pool
α_1	-0.267926	Transcription rate of g1
β	0.267926	$\frac{\delta}{K_1}$, δ =(degradation plus dilution) rate for gRNAs
P_{tot}	0.207065	Target promoter concentration
K	-0.128736	Association constant for dCas:gRNA complex binding to promoter.
α_x	-0.000000	Transcription rate of other gRNAs

Table 3.5: Sensitivity of scaling overperformance to each parameter.

α_x changes the concentration of competing guide RNA in the same way that increasing N does, while changing α_1 is equivalent to changing both α_x and C_{tot} . Therefore, changes to these two variables are roughly equivalent to shifting the performance curves shown in Figure 1 along the "# Competing Guides" axis, albeit in a nonlinear way.

Fold change overperformance is relatively unaffected by K and β , which both incorporate binding constants and are therefore parameters of particularly high uncertainty.

In contrast, scaling overperformance is fairly sensitive to β , though only somewhat more so than to P_{tot} (and less than to C_{tot} , which we can see from Figure 3.11 is still not particularly large (remember that overperformance is a *relative* measure of the effectiveness of CRISPRa vs. CRISPRi). We are therefore less confident in our ability to quantitatively predict scaling overperformance than fold change overperformance, but we believe the overall trend that CRISPRa performs better than CRISPRi under conditions of high gRNA competition will hold.

3.5 Engineering Requirements of CRISPRi

Although we do not yet have a complete set of guidelines for engineering arbitrary CRISPRi systems or combinations of systems, the models and simulations in

Sections 3.3 and 3.4 do provide a few key lessons:

- Active degradation of dCas can improve circuit performance, and may be necessary.
- Almost any amount of transcriptional leak can be quite destructive to CRISPRi circuits.
- The speed at which dCas:gRNA complexes bind to DNA is important for circuit function, and implementing CRISPRi circuits in vivo may require increasing this binding rate.
- CRISPRi circuits should fall off in performance with scale faster than CRISPRa circuits.

Each of these lessons is accompanied by an engineering requirement. Some possible strategies for fulfilling these requirements are outlined below.

Degradation of dCas

E. coli can actively degrade proteins using the ClpXP system, which uses the proteases ClpXP and ClpXA to selectively degrade proteins bearing *ssrA*, a small C-terminal peptide tag. The native ClpXP system degrades at least 400 times faster than the rate of cell dilution, which is far too fast for any of the circuits outlined in Section 3.3 (Farrell, Grossman, and Sauer, 2005). However, targeted mutations to the *ssrA* tag have been used to tune degradation rates to anywhere between 2 and 100 times the rate of dilution, which falls solidly into the target degradation range outlined in Figure 3.7 (Landry, Stöckel, and Pakrasi, 2013). Alternatively, dCas could be degraded using the *mf*-Lon protease system, which is similar to the ClpXP system in function and tunability but is orthogonal to any system in *E. coli* (Gur and Sauer, 2008; Cameron and J. Collins, 2014).

Improving fold-repression

The CRISPRi repressors reported in the literature typically repress with strengths between 10x and 100x. Moreover, the strongest CRISPRi repressors are typically targeted inside the target promoter, which limits their design space severely. Simulations suggest that leaks reported for elongation-blocking CRISPRi will break toggle switch and oscillator circuits unless degradation rates are extremely well-tuned. Engineering high fold-change in a promoter may be challenging, but several simple strategies are possible for improving repressor performance.

However, any “transcriptional leak” of the kind observed in any repression experiment can be explained as a consequence of slow DNA binding or weak binding equilibrium, rather than proper leak from bound promoters; in fact, real-world “leak” is likely to be a combination of the two. Our models predict that CRISPRi-lators without dCas degradation should only function if proper leak from dCas-bound promoters is extremely low.

Real-world CRISPRlators, though not particularly robust, do oscillate, suggesting that a consequence of our model would be that proper leak in CRISPRi systems is much smaller than leak from un-bound targets. If true, this suggests that efforts to reduce CRISPRi leak should be targeted at improving binding speed, rather than binding strength.

Faster dCas:DNA binding

As previously mentioned, the results shown in this report assume “best-case” binding rate of dCas:gRNA complexes to their target DNAs, with binding rates taken from in vitro association rate measurement. In live *E. coli*, dCas binding to DNA is much slower, with a single dCas complex estimated to require a few hours to bind to its target (Jones et al., 2017). A likely explanation for Cas9’s slow binding time is that dCas spends most of its time transiently bound to off-target PAM sites in the genome. The dCas protein can temporarily bind to any double-stranded DNA site with a correct PAM sequence. When it does, it briefly opens the DNA helix to “check” whether its associated guide RNA matches the sequence immediately adjacent to the PAM. Each non-target PAM present in the cell slows the rate of correct binding by acting as a low-affinity “decoy” binding site, which slows binding to the target promoter. The *S. pyogenes* Cas9 (by far the most widely-used Cas variant, and the one used exclusively in this report) uses the PAM "NGG", which can be expected to appear roughly 750,000 times in the (approximately diploid) genome of a growing *E. coli* cell. This represents substantial barrier to correct target identification.

There are several possible solutions to the problem of slow dCas:DNA binding in vivo. The most straightforward, at least conceptually, would be to move out of cells entirely and construct circuits exclusively in TX-TL or another cell-free system. Since cell-free systems do not have DNA replication or dilution to remove dCas from DNA, this would have to be done either in a microfluidic device capable of manually diluting a running reaction (see (Niederholtmeyer et al., 2015)) or using degradation-tagged dCas proteins.

Another simple way to speed up dCas:DNA binding rates would be to simply increase the concentration of either dCas (by increasing the baseline production of dCas) or target (by either genomically integrating multiple copies of the CRISPRi circuit or by expressing the circuit off of a plasmid). Simulations so far suggest that, all other things being equal, increasing the rate of dCas production has the effect of narrowing the window of acceptable dCas degradation rates. More simulation will be required to determine the feasibility of either of these two interventions.

Another possible solution would be to use a dCas (or another programmable binding protein) with a different, more complex PAM sequence. For example, the *Treponema denticola* Cas9 (TD-Cas) uses the PAM NAAAAC. After accounting for nonspecific PAM recognition, TD-Cas PAM sites ought to occur between 64 and 1024 times less frequently than *S. pyogenes* Cas PAM sites, with a corresponding increase in binding rate (Esvelt et al., 2013). If the *other* kinetics of TD-dCas are similar to those of *S. pyogenes* dCas, then we should expect TD-dCas binding to DNA to be

only perhaps twice as slow as in vitro dCas, which should be quite manageable.

Use CRISPRa where possible

There are good reasons to use CRISPRi instead of CRISPRa when building bacterial biocircuits, at least in the near future. CRISPRi requires fewer moving parts, produces larger best-case fold changes (and unquestionably higher fold changes in single-gRNA systems), and is less likely to drastically interfere with host genetic expression, at least in *E. coli*, as current-generation prokaryotic CRISPRa activators use modified versions of *E. coli* activators that upregulate some native promoters.

Repression is arguably also a more useful tool than activation. In particular, several simple, classic genetic circuits rely exclusively on repression (e.g., the repressilator (Elowitz and Leibler, 2000), the two-gene genetic toggle switch (Gardner, Cantor, and J. J. Collins, 2000), and NOR-gates (Gander et al., 2017)).

Finally, CRISPRa is more difficult to use on natural genomic targets, as it requires a PAM within a ~ 10 bp window of an ideal position upstream of the target promoter's -35 box. For some genes, this target simply does not exist.

Nevertheless, CRISPRa has a distinct advantage over CRISPRi—it should be significantly less impacted by bottlenecking of core dCas9. CRISPRi effectiveness is predicted to drop precipitously for systems above about a dozen gRNAs, while CRISPRa should still function in the presence of dozens-to-hundreds of competing gRNAs.

As CRISPR-based synthetic circuits grow in scale and complexity past about a dozen components, we anticipate that the usefulness of CRISPRi will drop off precipitously, while CRISPRa should still function even in the presence of dozens of competing gRNAs (at least, for circuits with low target concentrations). We urge anyone who dreams of building large CRISPR-based biocircuits to consider using CRISPRa.

3.6 Conclusions

CRISPRi remains an intriguing technology for scaling up genetic regulatory networks. However, building functional CRISPRi circuits is not as simple as sketching a repression net and targeting gRNAs against each other accordingly. In particular, ODE simulations of CRISPRi reveal specific functional requirements regarding dCas9 protein regulation and repressor characteristics. Some degradation of dCas may be required, but not *too* much; some leak from dCas-repressed promoters is acceptable, but not *too* much. Long timescales of DNA binding make CRISPRi construction more difficult, but not fatally so. Using these and other insights, it should be an achievable goal to build and express CRISPRi circuits, which would constitute an important milestone toward engineering cells with complex programmable behavior.

References

- Bikard, David, Wenyan Jiang, Poulami Samai, Ann Hochschild, Feng Zhang, and Luciano A. Marraffini (2013). “Programmable repression and activation of bacterial gene expression using an engineered CRISPR-Cas system”. In: *Nucleic Acids Research* 41.15, pp. 7429–7437. DOI: 10.1093/nar/gkt520.
- Boyle, Evan A., Johan O. L. Andreasson, Lauren M. Chircus, Samuel H. Sternberg, Michelle J. Wu, Chantal K. Guegler, Jennifer A. Doudna, and William J. Greenleaf (2017). “High-throughput biochemical profiling reveals sequence determinants of dCas9 off-target binding and unbinding”. In: *Proceedings of the National Academy of Sciences* 114.21, pp. 5461–5466. DOI: 10.1073/pnas.1700557114.
- Cameron, Ewen and James Collins (2014). “Tunable protein degradation in bacteria”. In: *Nature Biotechnology* 32.12, pp. 1276–1281. DOI: 10.1038/nbt.3053.
- Ceroni, Francesca, Rhys Algar, Guy-Bart Stan, and Tom Ellis (2015). “Quantifying cellular capacity identifies gene expression designs with reduced burden”. In: *Nature Methods* 12 (5). DOI: 10.1038/nmeth.3339.
- Chavez, Alejandro, Marcelle Tuttle, Benjamin W Pruitt, Ben Ewen-Campen, Raj Chari, Dmitry Ter-Ovanesyan, Sabina J Haque, Ryan J Cecchi, Emma J K Kowal, Joanna Buchthal, Benjamin E Housden, Norbert Perrimon, James J Collins, and George Church (2016). “Comparison of Cas9 activators in multiple species”. In: *Nature Methods* 13.7, pp. 563–567. DOI: 10.1038/nmeth.3871.
- Chen, Pin Yi, Yili Qian, and Domitilla Del Vecchio (2019). “A Model for Resource Competition in CRISPR-Mediated Gene Repression”. In: *Proceedings of the IEEE Conference on Decision and Control*. Vol. 2018-December, pp. 4333–4338. ISBN: 9781538613955. DOI: 10.1109/CDC.2018.8619016.
- Cho, Suhyung, Donghui Choe, Eunju Lee, Sun Chang Kim, Bernhard Palsson, and Byung Kwan Cho (2018). “High-Level dCas9 Expression Induces Abnormal Cell Morphology in Escherichia coli”. In: *ACS Synthetic Biology* 7.4, pp. 1085–1094. DOI: 10.1021/acssynbio.7b00462.
- Cui, Lun, Antoine Vigouroux, François Rousset, Hugo Varet, Varun Khanna, and David Bikard (2018). “A CRISPRi screen in E. coli reveals sequence-specific toxicity of dCas9”. In: *Nature Communications* 9.1. DOI: 10.1038/s41467-018-04209-5.
- Dong, Chen, Jason Fontana, Anika Patel, James M. Carothers, and Jesse G. Zalatan (2018). “Synthetic CRISPR-Cas gene activators for transcriptional reprogramming in bacteria”. In: *Nature Communications* 9.1. DOI: 10.1038/s41467-018-04901-6.
- Elowitz, Michael B. and Stanislas Leibler (2000). “A synthetic oscillatory network of transcriptional regulators”. In: *Nature* 403.6767, pp. 335–338. DOI: 10.1038/35002125.

- Esvelt, Kevin M, Prashant Mali, Jonathan L Braff, Mark Moosburner, Stephanie J Young, and George M Church (2013). “Orthogonal Cas9 proteins for RNA-guided gene regulation and editing”. In: *Nature Methods* 10.11, pp. 1116–1121. DOI: 10.1038/nmeth.2681.
- Farrell, Christopher M., Alan D. Grossman, and Robert Sauer (2005). “Cytoplasmic degradation of *ssrA*-tagged proteins”. In: *Molecular Microbiology* 57.6, pp. 1750–1761. DOI: 10.1111/j.1365-2958.2005.04798.x.
- Gander, Miles W., Justin D. Vrana, William E. Voje, James M. Carothers, and Eric Klavins (2017). “Digital logic circuits in yeast with CRISPR-dCas9 NOR gates”. In: *Nature Communications* 8, p. 15459. DOI: 10.1038/ncomms15459.
- Gardner, Timothy S., Charles R. Cantor, and James J. Collins (2000). “Construction of a genetic toggle switch in *Escherichia coli*”. In: *Nature* 403.6767, pp. 339–342. DOI: 10.1038/35002131.
- Gilbert, Luke, Matthew H Larson, Leonardo Morsut, Zairan Liu, Gloria A Brar, Sandra E Torres, Noam Stern-Ginossar, Onn Brandman, Evan H Whitehead, Jennifer A Doudna, Wendell A Lim, and Jonathan S Weissman (2013). “CRISPR-Mediated Modular RNA-Guided Regulation of Transcription in Eukaryotes”. In: *Cell* 154.2, pp. 442–451. DOI: 10.1016/j.cell.2013.06.044.
- Goentoro, Lea, Oren Shoval, Marc W. Kirschner, and Uri Alon (2009). “The Incoherent Feedforward Loop Can Provide Fold-Change Detection in Gene Regulation”. In: *Molecular Cell* 36.5, pp. 894–899. DOI: 10.1016/j.molcel.2009.11.018.
- Golubyatnikov, V. P. and V. V. Ivanov (2018). “Uniqueness and stability of a cycle in three-dimensional block-linear circular gene network models”. In: *Siberian Journal of Pure and Applied Mathematics* 18 (4), pp. 19–28. DOI: 10.33048/pam.2018.18.402.
- Gong, Shanzhong, Helen Hong Yu, Kenneth Johnson, and David Taylor (2017). “DNA unwinding is the primary determinant of CRISPR-Cas9 activity”. In: *Bioarxiv*. DOI: 10.1101/205823.
- Gur, Eyal and Robert Sauer (2008). “Evolution of the *ssrA* degradation tag in *Mycoplasma*: specificity switch to a different protease.” In: *Proceedings of the National Academy of Sciences of the United States of America* 105.42, pp. 16113–16118. DOI: 10.1073/pnas.0808802105.
- Ho, Hsing-I, Jennifer R Fang, Jacky Cheung, and Harris H Wang (2020). “Programmable CRISPR-Cas transcriptional activation in bacteria”. In: *Molecular Systems Biology* 16 (7). DOI: 10.15252/msb.20199427.
- Hsiao, Victoria, Yutaka Hori, Paul WK Rothmund, and Richard M Murray (2016). “A population-based temporal logic gate for timing and recording chemical events”. In: *Molecular Systems Biology* 12.5, p. 869. DOI: 10.15252/msb.20156663.

- Jinek, Martin, Krzysztof Chylinski, Ines Fonfara, Michael Hauer, Jennifer A. Doudna, and Emmanuelle Charpentier (2012). “A Programmable Dual-RNA – Guided”. In: *Science* 337, pp. 816–822. doi: 10.1126/science.1225829.
- Jones, Daniel, Cecilia Unoson, Prune Leroy, Vladimir Curic, and Johan Elf (2017). “Kinetics of dCas9 Target Search in Escherichia Coli”. In: *Biophysical Journal* 112.3, 314a. doi: 10.1016/j.bpj.2016.11.1700.
- Kim, Jongmin, Ishan Khetarpal, Shaunak Sen, and Richard M. Murray (2014). “Synthetic circuit for exact adaptation and fold-change detection”. In: *Nucleic Acids Research* 42.9, pp. 6078–6089. doi: 10.1093/nar/gku233.
- Kuo, James, Ruoshi Yuan, Carlos Sánchez, Johan Paulsson, and Pamela A Silver (2020). “Toward a translationally independent RNA-based synthetic oscillator using deactivated CRISPR-Cas”. In: *Nucleic Acids Research* 48 (14), pp. 8165–8177. doi: 10.1093/nar/gkaa557.
- Landry, Brian P., Jana Stöckel, and Himadri B. Pakrasi (2013). “Use of degradation tags to control protein levels in the cyanobacterium *Synechocystis* sp. strain PCC 6803”. In: *Applied and Environmental Microbiology* 79.8, pp. 2833–2835. doi: 10.1128/AEM.03741-12.
- Likhoshvai, Vitaly A., Vladimir P. Golubyatnikov, and Tamara M. Klebodarova (2020). “Limit cycles in models of circular gene networks regulated by negative feedback loops”. In: *BMC Bioinformatics* 21.255. doi: 10.1186/s12859-020-03598-z.
- Ma, Hanhui, Li-Chun Tu, Ardalan Naseri, Maximiliaan Huisman, Shaojie Zhang, David Grunwald, and Thoru Pederson (2016). “CRISPR-Cas9 nuclear dynamics and target recognition in living cells”. In: *The Journal of Cell Biology*, jcb.201604115. doi: 10.1083/jcb.201604115.
- Mali, Prashant, John Aach, P. Benjamin Stranges, Kevin M. Esvelt, Mark Moosburner, Sriram Kosuri, Luhan Yang, and George M. Church (2013). “CAS9 transcriptional activators for target specificity screening and paired nickases for cooperative genome engineering”. In: *Nature Biotechnology* 31.9, pp. 833–838. doi: 10.1038/nbt.2675.
- Mekler, Vladimir, Leonid Minakhin, Ekaterina Semenova, Konstantin Kuznedelov, and Konstantin Severinov (2016). “Kinetics of the CRISPR-Cas9 effector complex assembly and the role of 3’-terminal segment of guide RNA”. In: *Nucleic Acids Research* 44.6, pp. 2837–2845. doi: 10.1093/nar/gkw138.
- Niederholtmeyer, Henrike, Zachary Z. Sun, Yutaka Hori, Enoch Yeung, Amanda Verpoorte, Richard M. Murray, and Sebastian J. Maerkl (2015). “Rapid cell-free forward engineering of novel genetic ring oscillators”. In: *eLife* 4.OCTOBER2015. doi: 10.7554/eLife.09771.
- Nielsen, A. A. and C. A. Voigt (2014). “Multi-input CRISPR/Cas genetic circuits that interface host regulatory networks”. In: *Molecular Systems Biology* 10.11, pp. 763–763. doi: 10.15252/msb.20145735.

- Nielsen, Alec a K, Bryan S Der, Jonghyeon Shin, Prashant Vaidyanathan, Vanya Paralanov, Elizabeth a Strychalski, David Ross, Douglas Densmore, and Christopher a. Voigt (2016). “Genetic circuit design automation.” In: *Science (New York, N.Y.)* 352.6281, aac7341. doi: 10.1126/science.aac7341.
- Nissim, Lior, Samuel D. Perli, Alexandra Fridkin, Pablo Perez-Pinera, and Timothy K. Lu (2014). “Multiplexed and Programmable Regulation of Gene Networks with an Integrated RNA and CRISPR/Cas Toolkit in Human Cells”. In: *Molecular Cell* 54.4, pp. 698–710. doi: 10.1016/j.molcel.2014.04.022.
- Qi, Lei S., Matthew H. Larson, Luke A. Gilbert, Jennifer A. Doudna, Jonathan S. Weissman, Adam P. Arkin, and Wendell A. Lim (2013). “Repurposing CRISPR as an RNA-guided platform for sequence-specific control of gene expression”. In: *Cell* 152.5, pp. 1173–1183. doi: 10.1016/j.cell.2013.02.022.
- Rubens, Jacob R., Gianluca Selvaggio, and Timothy K. Lu (2016). “Synthetic mixed-signal computation in living cells”. In: *Nature Communications* 7, p. 11658. doi: 10.1038/ncomms11658.
- El-Samad, H El, D. Del Vecchio, and M Khammash (2005). “Repressilators and promotilators: loop dynamics in synthetic gene networks”. In: *Proceedings of the American Control Conference, 2005*. Pp. 4405–4410. doi: 10.1109/ACC.2005.1470689.
- Santos-Moreno, Javier, Eve Taisiudi, Joerg Stelling, and Yolonda Schaerlli (2020). “Multistable and dynamic CRISPRi-based synthetic circuits”. In: *Nature Communications* 11.2746. doi: 10.1038/s41467-020-16574-1.
- Santos-Moreno, Javier, Eve Tasiudi, Joerg Stelling, and Yolanda Schaerli (2020). “Multistable and dynamic CRISPRi-based synthetic circuits”. In: *Nature Communications* 11.2746. doi: 10.1038/s41467-020-16574-1.
- Shea, Madeline A. and Gary K. Ackers (1985). “The OR control system of bacteriophage lambda. A physical-chemical model for gene regulation”. In: *Journal of Molecular Biology* 181.2, pp. 211–230. doi: 10.1016/0022-2836(85)90086-5.
- Singh, Digvijay, Samuel H. Sternberg, Jingyi Fei, Jennifer A. Doudna, and Taekjip Ha (2016). “Real-time observation of DNA recognition and rejection by the RNA-guided endonuclease Cas9”. In: *Nature Communications* 7, p. 12778. doi: 10.1038/ncomms12778.
- Stricker, Jesse, Scott Cookson, Matthew R. Bennett, William H. Mather, Lev S. Tsimring, and Jeff Hasty (2008). “A fast, robust and tunable synthetic gene oscillator”. In: *Nature* 456.7221, pp. 516–519. doi: 10.1038/nature07389.
- Swaminathan, Anandh, Marcella Gomez, David Shis, Matthew R. Bennett, and Richard M. Murray (2016). “Stochastic Gene Expression in Single Gene Oscillator Variants”. In: *Synthetic Biology: Engineering, Evolution and Design (SEED) Conference*.

Zalatan, Jesse G., Michael E. Lee, Ricardo Almeida, Luke A. Gilbert, Evan H. Whitehead, Marie La Russa, Jordan C. Tsai, Jonathan S. Weissman, John E. Dueber, Lei S. Qi, and Wendell A. Lim (2015). “Engineering complex synthetic transcriptional programs with CRISPR RNA scaffolds”. In: *Cell* 160.1-2, pp. 339–350. DOI: [10.1016/j.cell.2014.11.052](https://doi.org/10.1016/j.cell.2014.11.052).

Zhang, Shuyi and Christopher A. Voigt (2018). “Engineered dCas9 with reduced toxicity in bacteria: implications for genetic circuit design”. In: *Nucleic acids research* 46.20, pp. 11115–11125. DOI: [10.1093/nar/gky884](https://doi.org/10.1093/nar/gky884).

Chapter 4

HOW TO MODEL REPLICATING DNA (AND WHY)

4.1 Introduction

Although most synthetic biocircuits use DNA, models of synthetic biocircuits typically do not explicitly describe the dynamics of those DNA species. For example, the original repressilator model tracked mRNA and protein species, but not DNA (Elowitz and Leibler, 2000), and the model for the first genetic toggle switch simply tracked “repressor 1” and “repressor 2” (Gardner, Cantor, and Collins, 2000). For many circuits under many modeling assumptions, it is sufficient to assume that all DNA species are held at a fixed concentration by the cell, by means of mysterious machinery whose details are irrelevant to understanding the circuit.

In some cases, however, it is useful or necessary to explicitly represent DNA as a dynamic species. Some circuits, for example, use DNA in different states as a dynamic component or readout (for example, integrase-based state machines (Roquet et al., 2016)). It can also be useful to explicitly represent DNA when DNA binding is slow relative to the other circuit processes (for example, CRISPR-based transcription factor networks under some conditions (Jones et al., 2017)). Stochastic models, in particular, are often most naturally expressed using explicit DNA species. We will primarily deal with the case of a plasmid of reasonably high copy number, for which we should be able to largely neglect the possibility of plasmid loss by copy number fluctuation.

Explicitly-modeled DNA often requires some mechanism of replication—again particularly in stochastic models. Unfortunately, the obvious replication implementation $DNA \rightarrow DNA + DNA$ is a trap that leads to pathological circuit behavior.

4.2 Trivially-Replicating Plasmids Have No Well-Defined Copy Number

An obvious way to model replication of a DNA species is the straightforward



shown visually in Figure 4.1A. Here we use a non-indexed DNA representing a single plasmid or genome, which may contain many functional sub-units (or none at all). We refer to this simple replication mechanism as “trivial self-replication.” Although trivial replication is intuitively appealing, we strongly recommend against its use.

First, consider trivial self-replication in the mass-action, high-concentration regime (we will consider stochastic dynamics shortly). Coupled with a dilution reaction $DNA \xrightarrow{\gamma} \emptyset$, trivially self-replicating DNA has dynamics

$$\frac{dDNA}{dt} = (\alpha - \gamma)DNA.$$

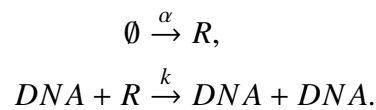
Notice that at steady state, we have $\alpha - \gamma = 0$, *independent of DNA concentration*. In other words, the only finite, non-zero steady state of the trivial replication mechanism occurs when production is precisely balanced by dilution, and such a steady state is structurally unstable. If production is slightly faster than dilution, then *DNA*'s concentration will explode unphysically (and unbiologically) to infinity. Conversely, if dilution is slightly faster than production, *DNA* will always fall to zero concentration and die out.

In a deterministic models of a trivially-replicating plasmid, production and dilution *can* be balanced perfectly, giving a nominally constant concentration of *DNA*, but this mechanism of replication rejects no disturbances—any addition or removal of *DNA* will remain permanently uncorrected.

Stochastic simulation of the trivial model cannot even achieve this level of marginal stability. Stochastic simulation is, by its nature, noisy; a stochastically simulated, trivially replicating *DNA* will random walk in copy number until, practically speaking, it either dies out by wandering to zero or explodes to a concentration too large to simulate (shown in Figure 4.1A).

4.3 Zero-Order Replication Recovers Good Steady-State Properties

The trivial replication mechanism is unstable because it makes both production and degradation of *DNA* linearly dependent on the concentration of *DNA* itself. A simple way to add stability to the replication model is to remove that dependence by conditioning replication on a “dummy replication trigger” produced at a constant rate (shown diagrammatically in Figure 4.1B):



This mechanism has ODE dynamics

$$\begin{aligned} \frac{dR}{dt} &= \alpha - kR * DNA \\ \frac{dDNA}{dt} &= kR * DNA - \gamma DNA. \end{aligned}$$

At steady state, $R = \frac{\alpha}{k * DNA}$, which cancels out *DNA* in *DNA*'s production term and gives $DNA = \frac{\alpha}{\gamma}$. This steady state is stable. As long as *k* is fast relative to

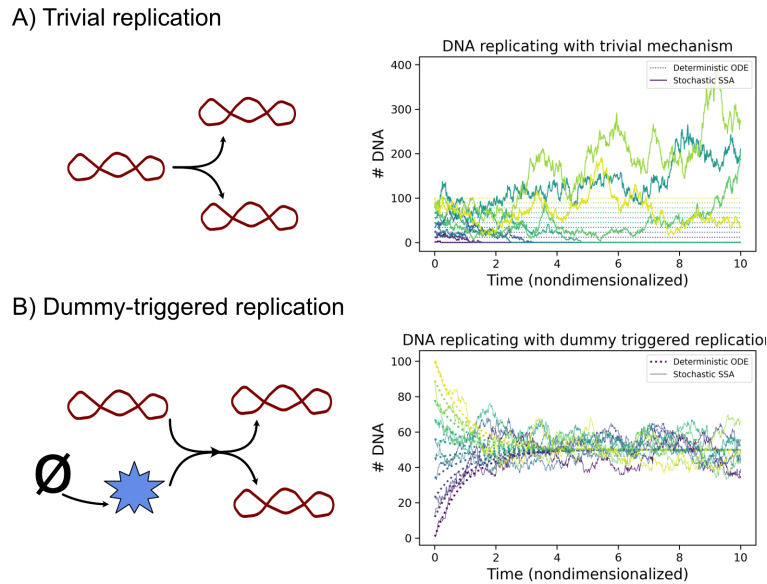


Figure 4.1: **a)** The trivial replication mechanism, in which *DNA* spontaneously self-replicates. This mechanism is unstable and produces random-walking *DNA* concentrations. **b)** The dummy-triggered replication mechanism, in which replication is triggered by a dummy molecule produced at a constant rate. This mechanism, coupled to dilution, is stable and rejects small disturbances.

other replication dynamics (α and γ), the dummy-triggered replication mechanism emulates zero-order replication of *DNA*.

Dummy-triggered replication leads to a stable steady state concentration of *DNA*, as shown in Figure 4.1B. We therefore recommend them over the trivial replication mechanism.

4.4 Biological Models of Plasmid Replication Approximately Reduce to Zero-Order Replication

Dummy-triggered replication is not meant to accurately describe a real biological processes. Real cells do not control replication using consumable molecules like *R*. The critical property of the dummy-triggered replication mechanism is its ability to achieve the (biologically important!) property of having a defined, stable steady state.

Nevertheless, we will show that under a few relatively mild assumptions, at least one real-world *DNA* replication mechanism can be reduced to a zero-order replication mechanism equivalent to the dummy-triggered replication mechanism, .

Consider the ColE1 plasmid replication system, first crystallized mathematically by Brendel and Perelson in 1993, shown on the left in Figure 4.2. We use ColE1 as an example because it has a particularly simple and well-understood replication mechanism.

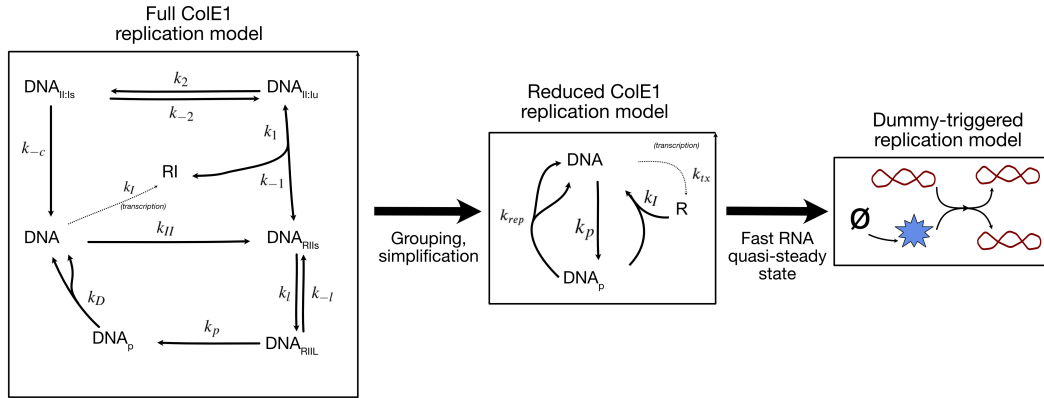


Figure 4.2: A mechanistic model of ColE1 (left), which can be conceptually approximated by a simplified ColE1 model (middle), which under fast RNA dynamics reduces zero-order replication equivalent to that of the dummy-triggered replication model (right).

Briefly, ColE1 plasmids replicate using a *cis*-acting RNA primer, called RNAII. That primer can be blocked by a complementary RNA called RNAI, which is constitutively produced by the ColE1 plasmid. This means that the per-plasmid rate of ColE1 replication drops as the concentration of ColE1 plasmids increases, which gives ColE1 a finite steady state.

In the absence of RNAI, a ColE1-based DNA species are duplicated by initiation of RNAII transcription ($DNA \rightarrow DNA_{RIIS}$), elongation of RNAII ($DNA_{RIIS} \rightarrow DNA_{RIIL}$), attachment of a DNA polymerase to the RNAII-primed DNA complex ($DNA_{RIIL} \rightarrow DNA_P$), and, finally, replication by the attached DNA polymerase ($DNA_P \rightarrow DNA + DNA$). If a DNA polymerase does not bind to an elongated RNAII, the RNAII will be cleaved off, returning the plasmid to its initial state ($DNA_{RIIL} \rightarrow DNA$).

ColE1 does, however, produce RNAI ($DNA \rightarrow DNA + RI$). RNAI can bind to RNAII while it's being transcribed, blocking elongation ($DNA_{RIIS} + RI \rightarrow DNA_{II:IU}$). The RNAI:RNAII complex formed this way is unstable and can reverse easily, but can switch into a much more stable form ($DNA_{II:IU} \rightarrow DNA_{II:IS}$) from which both RNAs can be cleaved off the plasmid by RNase action, resetting the plasmid ($DNA_{II:IS} \rightarrow DNA$). This serves as the negative feedback mechanism that lowers ColE1's replication rate as its copy number increases.

In their original description of this model, Brendel and Perelson also describe optional sequestration of the $DNA_{II:IS}$ state by *Rom* binding, which leads to a lower steady state copy number (Brendel and Perelson, 1993). Freudenau et al further extend this model with additional feedback by uncharged tRNA, which occurs under starvation conditions, and experimentally parameterize the model. For simplicity, I omit these two extensions in this work.

Although the details of Brendel and Perelson's model are a bit messy, they implement

a straightforward logic:

- *DNA* produces an RNA species *RI*. More *DNA* leads to more *RI*.
- Occasionally, *DNA* will spontaneously enter a primed state from which it can replicate. (Biologically, the plasmid produces an RNA primer *RNAII* which can initiate replication.)
- *RI* can react with a replication-primed *DNA*, un-priming it and destroying the *RI*. (*RI* binds to *RII* to form an inert complex that is removed by RNases.)
- If not stopped by an *RI*, a primed *DNA* can spontaneously replicate into two non-primed *DNA*. (DNA polymerase initiates replication using *RII* as a primer.)

We can represent this logic more clearly with a three-species reduction of Brendel and Perelson's model, shown in the middle in Figure 4.2, consisting of “unprimed” *DNA*, “primed” *DNA_p*, and feedback RNA *R*.

Under realistic parameter regimes (specifically, when *R* dynamics are fast and k_p is relatively fast compared to k_{rep}), this three-species ColE1 model can be approximated by a single-species model equivalent to the dummy-triggered replication model with replication rate $k_{rep}k_p(\gamma + \gamma_I)/(k_I(k_{tx} - k_p))$.

This approximation requires:

1. Dynamics of creation and destruction of *R* must be fast (i.e., *R* must be at quasi-steady state compared to *DNA* and *DNA_p*).
2. Replication must be bottlenecked by the $DNA_p \rightarrow DNA + DNA$ reaction (i.e., $k_p \gg k_{rep}$). This is true for the three-species model with parameters fit against simulations from the full ColE1 model.
3. $k_{tx} > k_p > \gamma$ and $k_{rep} > \gamma$ (required for stability of the three-species ColE1 model).

Since *R* dynamics are fast, we'll assume that *R* is at quasi-steady state:

$$\frac{dR}{dt} = k_{tx}DNA - (\gamma + \gamma_I)R - k_IDNA_pR$$

$$\text{At steady state: } R = \frac{k_{tx}DNA}{(\gamma + \gamma_I) + k_IDNA_p}$$

Because k_{rep} is, by assumption, relatively slow, we can also consider the quasi-steady state concentration of *DNA_p*. At quasi-steady state, the equilibrium condition between *DNA* and *DNA_p* means that

$$k_I * DNA_p * \frac{k_{tx}DNA}{(\gamma + \gamma_I) + k_I DNA_p} = k_p * DNA,$$

so

$$DNA_p = \frac{k_p(\gamma + \gamma_I)}{k_I(k_{tx} - k_p)}.$$

Surprisingly, we have discovered that the concentration of DNA_p is roughly *constant* regardless of the copy number of the plasmid. As long as this is true, the rate of replication (i.e., the rate at which DNA_p goes to $2DNA$) is

$$k_p * DNA_p = \frac{k_{rep}k_p(\gamma + \gamma_I)}{k_I(k_{tx} - k_p)}.$$

Since this is constant, we are back to the zero-order (e.g., dummy-triggered) replication model where DNA species replicate at a constant, DNA -independent rate! This zero-order, one-species reduced model will bring DNA to the same copy number and usually has behavior qualitatively very similar to the three-species model (Figure 4.3).

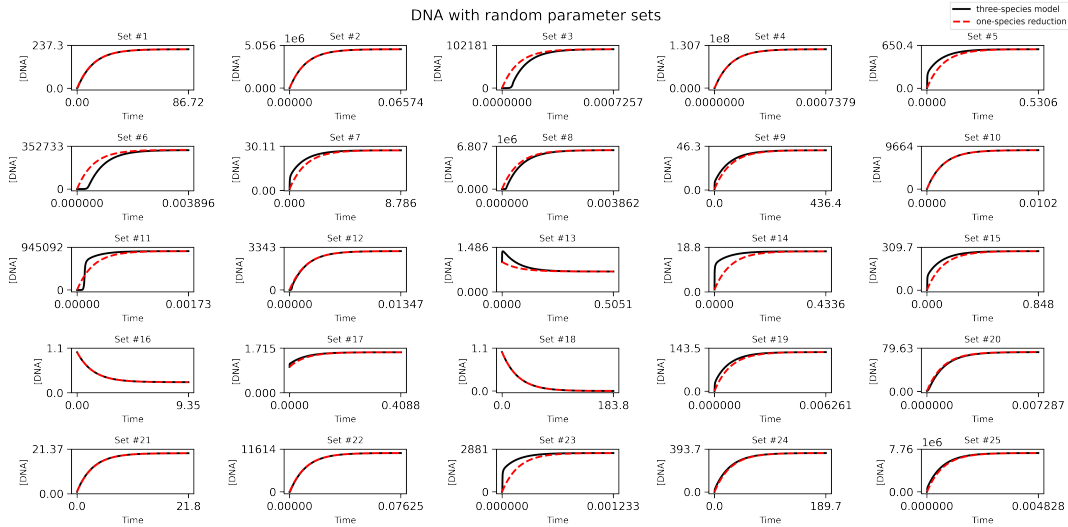


Figure 4.3: Simulated trajectories of replicating DNA the three-species ColE1 model (black lines), with trajectories for the equivalent one-species reduced model (dashed red lines), for 25 random parameter sets chosen so that DNA has a finite, nonzero steady state and $k_p > k_{rep}$.

Note that in the opposite regime, where $k_{rep} \gg k_p$, a similar reduction can be made to a single-species model where DNA replicates at rate

$$\frac{k_p}{1 + \beta DNA}, \text{ where } \beta = \frac{k_I k_{IX}}{k_{rep}(\gamma + \gamma_I)}.$$

4.5 Worked Examples

The CRISPRlator

One example of a circuit that can benefit from explicitly representing replicating DNA is the CRISPRlator from Chapter 3 (Figure 4.4).

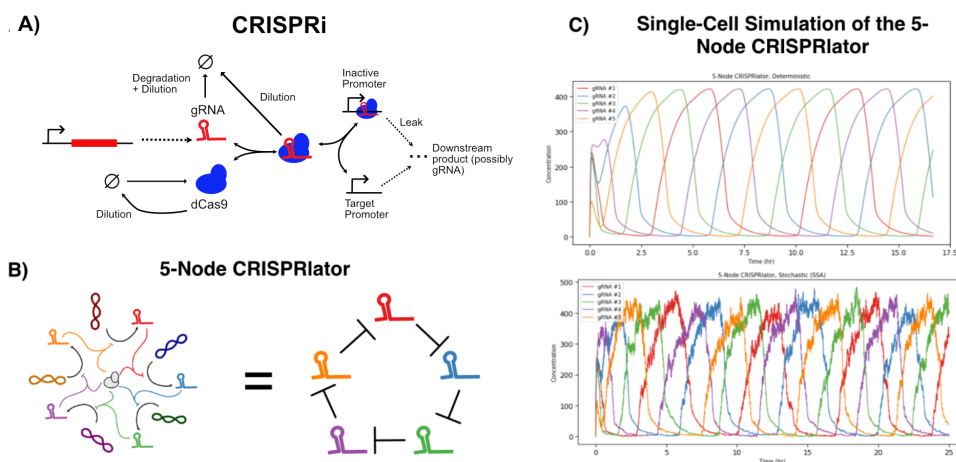


Figure 4.4: **a)** A simple model of general CRISPRi systems. Guide RNAs (gRNAs) complex with a shared pool of dCas9. dCas9:gRNA complexes bind to DNA targets, blocking transcription from those targets. Dashed arrows represent catalytic production (transcription). Targets can be other gRNAs, as in the 5-node CRISPRi oscillator (CRISPRlator) **(b)**. **c)** Deterministic and stochastic (SSA, non-lineage) simulation of the 5-node CRISPRlator.

We have already noted that dCas9 has shockingly slow binding kinetics—a single dCas9 molecule has been calculated to take an average of *six hours* to find a single DNA target in an *E. coli* cell, necessitating large numbers of targets and/or dCas9 molecules for circuits to operate on reasonable timescales (Jones et al., 2017). Thus, we have opted to dispose of the usual Hill function approximation of gene repression (one of the most common simplifying assumptions in the gene circuit literature) because that assumption *assumes* fast binding kinetics. To see the effects of slow binding kinetics, we have to explicitly model slow binding kinetics, which means explicitly tracking of bound and unbound DNA species. DNA replication might be a nice feature in such a model.

We might also want to probe how fluctuations in plasmid copy number affect the performance of the CRISPRlator. Can the CRISPRlator function if DNA copy numbers fluctuate in ways consistent with noisy DNA replication? Can it survive

random partitioning of plasmids during cell division? Will a CRISPRlator stay synchronized across a population of growing cells? Simulations with DNA replication can help answer those questions.

A few problems will immediately pop up if we try to simulate a CRISPRlator in a lineage of growing, dividing cells. The first is that if a cell keeps a constant *copy number* of a plasmid while *growing in volume*, then the concentration of plasmid will consistently and unrealistically drop by a factor of two with every growth cycle. We also run into a thorny question of how to add more plasmids to cells when they divide. The obvious thing to do is to instantaneously replicate all of the plasmids at cell division time, but this will instantaneously create many un-repressed plasmids, which can scramble the state of the CRISPRlator and kill oscillations.

A simpler solution is to add self-replication to the plasmid bearing the CRISPRlator. Figure 4.5 shows representative simulations of a single-cell 5-node CRISPRlator using each DNA replication method.

More interestingly, we can now simulate lengthy lineages of growing, replicating cells expressing the CRISPRlator. Figure 4.5E shows results from a lineage simulation using the dummy-triggered replication model parameterized identically to the one used for Figure 4.5B. This simulation begins with a single cell, which grows and divides every doubling of cell size. The total population is capped at 64 by killing a random cell whenever a cell division would bring the population above 64. Each dot is the concentration of total gRNA for one of the five gRNA in one cell at one time. These simulations show that over 150 generations, the CRISPRlator remains largely coherent, although increasingly less so as the simulation progresses (note the spread in the “tails” where each gRNA species falls to zero concentration).

Finally, we can simulate the effect of dropping down the copy number of the plasmids bearing the CRISPRlator. Figure 4.6 shows representative traces from one of the five gRNAs in CRISPRlator variants on plasmids varying from single-copy to copy number ten. Guide RNA production rates are scaled so that the expected gRNA production rate are the same as in the simulations used for Figure 4.5, and selection has been added to the models to counteract plasmid loss.

A single-cell temporal logic circuit

As a second case study, consider the integrase-based temporal logic circuit (Hsiao et al., 2016), which is a special case of the general recombinase-based state machine (one implementation of which is given by Roquet et al., 2016). This circuit uses two different serine DNA integrases *intA* and *intB* under inducible control of molecules *A* and *B*, respectively, to flip pieces of a shared reporter module. Induction of either integrase permanently alters the reporter module so that the circuit produces different outputs depending on whether it has been exposed to *A* only, *B* only, *A* followed by *B* ($A \Rightarrow B$), *B* followed by *A* ($B \Rightarrow A$), or no inducer at all.

A single cell expressing a temporal logic circuit is an imperfect sensor. Even if the cell is exposed to $A \Rightarrow B$, it is possible that it could, by stochastic fluctuation, not

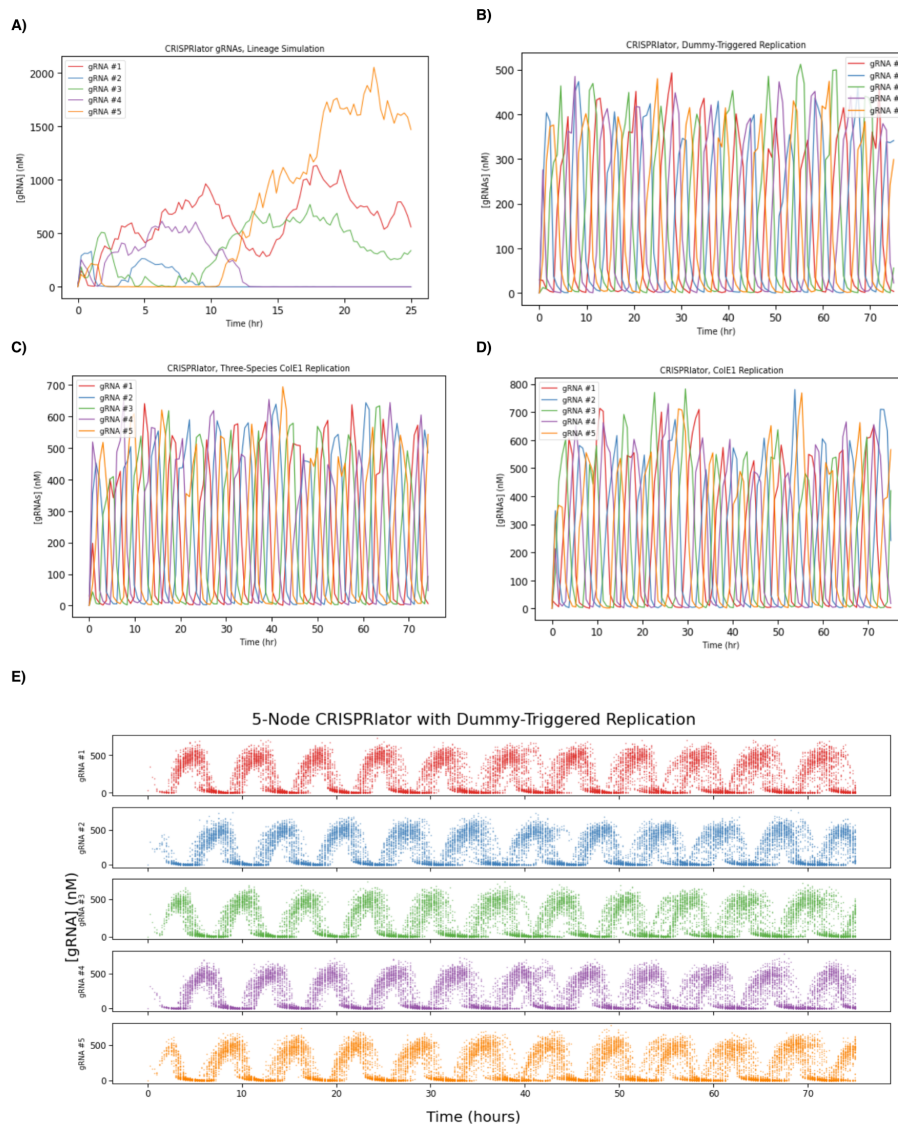


Figure 4.5: The five-node CRISPRiator in a single growing cell with DNA replication modeled using (a) the trivial replication model (note the shorter time scale), (b) dummy-triggered replication, (c) Brendel and Perelson's ColE1 replication model, and (textbfd) the simplified 3-species ColE1 replication model. Each line is a sum of the *concentrations* of all species containing one of the five gRNAs. (e) Lineage simulation of the 5-node CRISPRiator in growing, dividing cells over approximately 150 generations, with a population cap of 64 cells.

flip with integrase *intA* for so long that it eventually runs into a *B* molecule and integrates first with *intB*. The shorter the time delay between the introduction of *A* and *B*, the more likely this sort of mis-firing will be.

In a large population of temporal-logic-circuit bearing cells, this imperfection can be exploited to create a population-level, analog measurement of not just which input

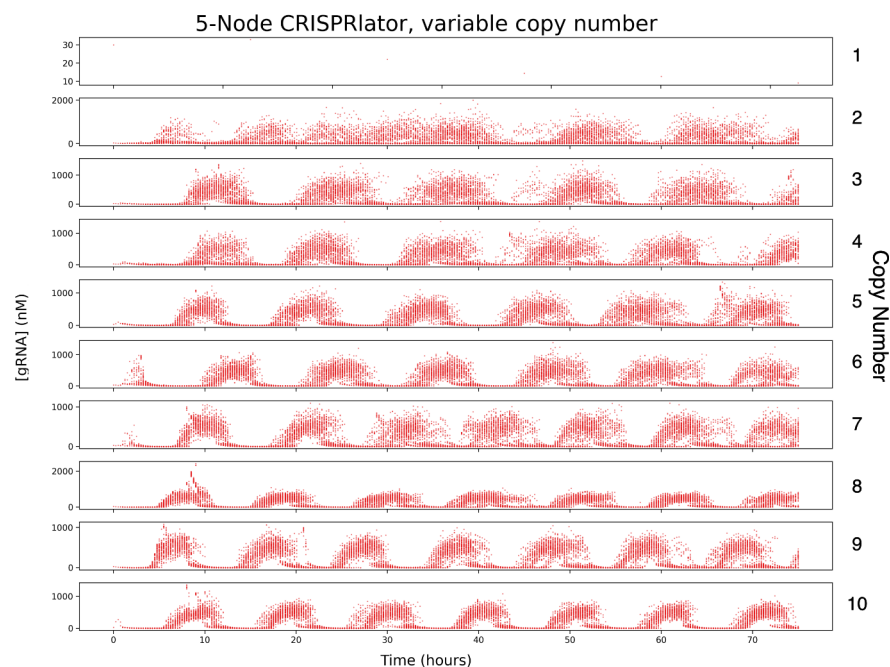


Figure 4.6: 5-node CRISPRlator at different copy numbers.

appeared first, but the *time delay* between the appearance of the two. If one input appears rapidly after the other, then many cells will “misfire” and report the wrong temporal value; if the two inputs are separated by a significant amount of time, then almost all cells will react to the first input before the second input arrives, and the population will homogeneously report the correct value. With proper calibration, the final distribution of cell states can be used to back out the time difference between inputs.

Equivalently, this type of population-level circuit can be used to measure differences in *concentration* of inputs that appear at the same time.

Importantly, the population-level circuit assumes that each cell has a single, simple state. This is achieved by using a chromosomally-integrated reporter module, so that the cell has a single, unique state (or, at least, will after a round or two of division). One could easily imagine performing the same type of population-level measurement in a single cell by using a state-bearing plasmid with a high copy number. This could make the circuit much more compact, and potentially simpler to sample depending on the circuit’s intended environment.

Will a single-cell, plasmid-based temporal logic gate still perform as advertised? There are practical, integrase-related difficulties with making such a circuit—as originally designed, multiple copies of the gate’s reporter module would recombine in unexpected and unwanted ways—but even setting those aside, moving a population-level circuit into a single cell adds new complications and sources of noise. For one thing, recording events between modules are no longer independent—transcription

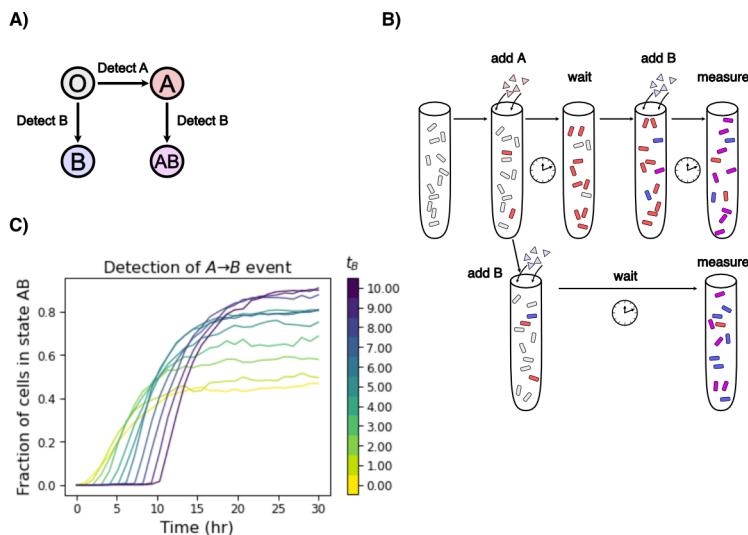


Figure 4.7: Overview of the temporal logic gate. **(A)** State machine describing a single temporal logic gate unit. “Detection” is classically implemented with integrases whose activities are induced by signals *A* and *B*. **(B)** The behavior of a population of cells where each cell contains a single temporal logic gate. The final distribution of cell states can be used as a readout of the time delay between introduction of *A* and introduction of *B*. **(C)** Representative simulation of a genomically-integrated temporal logic gate (i.e., without explicitly replicating DNA species). Each curve is the fraction of cells in state *AB* over time for a different delay time t_B between introduction of *A* and *B*.

of a single integrase mRNA can produce a number of module-flipping events. The frequency of a plasmid state in a cell might also not be stable over generations, due to random partitioning at division.

A straightforward way to ask whether or not a single-cell temporal logic gate can still give analog measurements is to simulate it. To do so, we should use stochastic simulation (since stochastic fluctuations are likely important for plasmid dynamics) and we will need to model plasmid replication (since stateful plasmids are the principle species acted on by the circuit).

When we add plasmid replication to the temporal logic gate, we can see that it still functions on a replicating plasmid, although with more noise than the original genome-located temporal logic gate (Figure 4.8)—but this version only requires a single cell, where the original was designed to work in a population of thousands to millions of cells.

4.6 Discussion

This work is far from the first to consider models of plasmid replication, and many modelers have crafted their own solutions to the problems that crop up when modeling replicating DNAs. There is not, however, any widespread consensus on

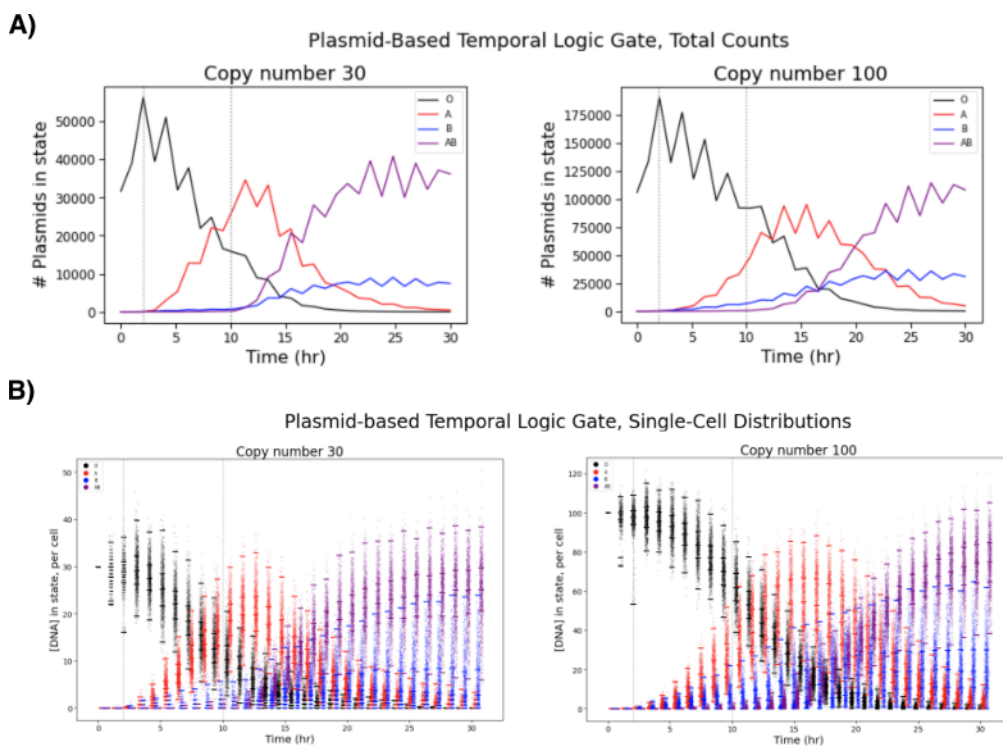


Figure 4.8: The temporal logic gate on a replicating plasmid. **(a)** Total counts of plasmids of copy number 30 or 100 in each state across a population of 1,056 cells introduced to *A* at time 2 hours and *B* at time 10 hours. **(b)** Same data as in **(a)**, disaggregated into individual cells' copy numbers of plasmids in each state. Points are shifted and jittered slightly on the time axis for visibility. Solid ticks mark the 1st, 25th, 50th, 75th, and 99th percentile copy numbers for each state at each time.

how to handle DNA replication in the context of synthetic biocircuit modeling.

One reason such a gap exists is that biocircuit modeling is rarely performed in the stochastic regime. In a deterministic ODE model, trivial DNA replication can be balanced perfectly against dilution. Such a trivially-replicating plasmid would be merely neutrally-stable, which is sufficient so long as no perturbations are applied to total DNA concentration. It is not clear that any more sophisticated replication mechanism is really required for deterministic models.

Once we move to the world of stochastic simulations (as predictive modelers ought to do, if their computational resources are adequate to the task!), trivial replication fails spectacularly, and the dummy-triggered mechanism (or something similar) becomes absolutely necessary.

Another possible benefit of dummy-triggered replication is that it does not hide itself. When trivial replication is used in deterministic models, the terms representing DNA replication in the model's ODEs tend to cancel out entirely, leaving no record of the replication mechanism in the system's mathematical description. In practice, replication is often also dropped from a model's CRN description. From

the perspective of a scientist or engineer studying a single model, this disappearing act may appear advantageous as it “cleans up” the relevant math, but this strategy has allowed DNA replication mechanisms to go largely undiscussed and unnoticed in the synthetic biology literature.

It is our hope that this document can serve as a reference guide for modelers first encountering DNA replication, at least until the field develops better standard practices around the subject.

References

- Brendel, V and A S Perelson (1993). “Quantitative model of ColE1 plasmid copy number control”. In: *Journal of Molecular Biology* 4 (299), pp. 860–872. DOI: 10.1006/jmbi.1993.1092.
- Elowitz, Michael B. and Stanislas Leibler (2000). “A synthetic oscillatory network of transcriptional regulators”. In: *Nature* 403.6767, pp. 335–338. DOI: 10.1038/35002125.
- Gardner, Timothy S., Charles R. Cantor, and James J. Collins (2000). “Construction of a genetic toggle switch in *Escherichia coli*”. In: *Nature* 403.6767, pp. 339–342. DOI: 10.1038/35002131.
- Hsiao, Victoria, Yutaka Hori, Paul WK Rothemund, and Richard M Murray (2016). “A population-based temporal logic gate for timing and recording chemical events”. In: *Molecular Systems Biology* 869.12. DOI: 10.15252/msb.20156663.
- Jones, Daniel, Cecilia Unoson, Prune Leroy, Vladimir Curic, and Johan Elf (2017). “Kinetics of dCas9 Target Search in *Escherichia coli*”. In: *Biophysical Journal* 112.3, 314a. DOI: 10.1016/j.bpj.2016.11.1700.
- Roquet, Nathaniel, Ava P. Soleimany, Alyssa C. Ferris, Scott Aaronson, and Timothy K. Lu (2016). “Synthetic recombinase-based state machines in living cells”. In: *Science* 353.aad8559 (6297). DOI: DOI:10.1126/science.aad8559.

CONCLUSIONS AND FUTURE WORK

5.1 Programming and Simulating Chemical Reaction Networks on a Surface

We have only scratched the surface of what a surface CRN can do. We have suggested a number of specific problems and exercises for the reader in Chapter 2; much more open-ended is the basic question, *what should we design next?* We have shown that synchronous cellular automata can be built with a surface CRN; what such automata would be most useful to construct? We have shown that surface CRNs are capable of both primitive and sophisticated pattern-manufacture; how do we increase the scale and complexity of surface CRN manufacturing in an efficient way? We have shown that surface CRNs are capable of simple swarm robot behavior; how much behavioral complexity can be efficiently packed into a surface CRN?

There also remains the practical challenge of *physically implementing a surface CRN*. To anyone with passing familiarity with chemistry, the idea of designing custom molecules that undergo specific, tunable reactions may seem difficult in the extreme, but DNA nanotechnology can get us surprisingly close to just that even today. DNA origami tiles with loose single strands as functionalization sites provide us with a simple and semi-scalable programmable surface on which to precisely place DNA complexes, which themselves can be programmed with sequence-targeted binding and strand displacement reactions. We may not currently have a fully programmable surface CRN breadboard, but the technology required to build one likely already exists.

We firmly believe the most important and revelatory applications of the surface CRN may be ones that have yet to be conceived, and we hope that this work serves as a gateway to a much vaster realm of molecular design.

5.2 Modeling Dynamic Transcriptional Circuits with CRISPRi

A fundamental problem we have encountered repeatedly in our modeling of CRISPRi is that parameters matter, and our knowledge of most biological rate parameters is thin at best. Our knowledge of the parameters of CRISPRi is better than that of most transcription factors in the literature, but there remains a great deal of uncertainty about how CRISPRi actually behaves in real cells.

That said, at least two different physical realizations of the CRISPRlator analyzed in Chapter 3 have been built to date, so we can now start to compare our models of CRISPRi against real circuit performance (Santos-Moreno et al., 2020; Kuo et al., 2020). So far, those models are somewhat informative but only partially capture the complexity of real CRISPRlator behavior. Even stochastic simulations of the CRISPRlator do not consistently display as much variability in pulse timing, duration, or amplitude as real implementations, and physically implemented

CRISPRlators appear to be consistently slower than their simulated counterparts. Elucidating the causes of these differences would help make existing CRISPRi models more useful design tools.

Furthermore, the dCas9 CRISPRlator of (Kuo et al., 2020) turned out to be largely inferior to a similar CRISPRlator built from a deactivated version of Cas12a. It may well turn out that future CRISPRi work will be done with Cas12a, another Cas-family protein, or another CRISPR component altogether; in any of these cases, we would do well to expand our CRISPR models.

5.3 How to Model Replicating DNA (and why)

We argue that the replication mechanism proposed in Chapter 4 is sufficient for most modelers' needs, and has some biological plausibility. It is not, however, a detailed mechanistic model, and may well generate unrealistic plasmid distributions (though still far better than those of the trivial replication mechanism).

Shao et al., 2021 measure copy number distributions empirically using fluorescent transcriptional repressors that bind to tandem arrays on the plasmid (though we would caution that plasmid copy number measurement is difficult and fraught with hard-to-detect biases (Tal and Paulsson, 2012)). The dummy-triggered replication model is unable to replicate those empirical distributions—the model only has a one (non-dimensionalized) degree of freedom in its parameters, and when that parameter is tuned so that our model's mean matches any empirically-observed mean, the model predicts a distribution with far lower variance than the empirically-measured distribution.

Interestingly, Brendel and Perelson's ColE1 model, used with their suggested parameters, agrees far more closely with the dummy-triggered replication model than with empirical measurements, and in practice it has proven difficult to independently tune the mean and variance of copy number distributions produced with the ColE1 model. Rigorous parameter inference on the ColE1 model would be a straightforward way to determine whether or not that model is capable of matching empirical copy number observations at all. Inference on stochastic models (required for distribution measurements) is non-trivial, but may be possible using noise-tolerant optimization algorithms such as simultaneous perturbation stochastic approximation (SPSA) (Spall, 1998).

References

- Kuo, James, Ruoshi Yuan, Carlos Sánchez, Johan Paulsson, and Pamela A Silver (2020). "Toward a translationally independent RNA-based synthetic oscillator using deactivated CRISPR-Cas". In: *Nucleic Acids Research* 48 (14), pp. 8165–8177. DOI: [10.1093/nar/gkaa557](https://doi.org/10.1093/nar/gkaa557).
- Santos-Moreno, Javier, Eve Taisiudi, Joerg Stelling, and Yolonda Schaerlli (2020). "Multistable and dynamic CRISPRi-based synthetic circuits". In: *Nature Communications* 11.2746. DOI: [10.1038/s41467-020-16574-1](https://doi.org/10.1038/s41467-020-16574-1).

- Shao, Bin, Jayan Rammohan, Daniel A. Anderson, Nina Alperovich, David Ross, and Christopher A. Voigt (2021). “Single-cell measurement of plasmid copy number and promoter activity”. In: *Nature Communications* 1475.12. DOI: 10.1038/s41467-021-21734-y.
- Spall, J. C. (1998). “Implementation of the simultaneous perturbation algorithm for stochastic optimization”. In: *IEEE Transactions on Aerospace and Electronic Systems* 34 (3), pp. 817–823. DOI: 10.1109/7.705889.
- Tal, Shay and Johan Paulsson (2012). “Evaluating quantitative methods for measuring plasmid copy numbers in single cells.” In: *Plasmid* 67.2, pp. 167–73. DOI: 10.1016/j.plasmid.2012.01.004.