

**Molecules Computing:
Self-Assembled Nanostructures, Molecular Automata,
and Chemical Reaction Networks**

Thesis by
David Soloveichik

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy



California Institute of Technology
Pasadena, California

2008
(Defended May 5, 2008)

Contents

Acknowledgements	vi
Abstract	vii
1 Introduction	1
1.1 The Promise of Bionanotechnology	1
1.2 The Mathematics of Computer Science	2
1.3 The Criteria for Success	3
1.4 The Contributions	3
1.4.1 Tile Self-Assembly	3
1.4.2 Restriction Enzyme Automata	4
1.4.3 Chemical Reaction Networks	4
Bibliography	6
2 Complexity of Self-Assembled Shapes	8
2.1 Abstract	8
2.2 Introduction	8
2.3 The Tile Assembly Model	9
2.3.1 Guaranteeing Unique Production	10
2.4 Arbitrarily Scaled Shapes and Their Complexity	11
2.5 Relating Tile-Complexity and Kolmogorov Complexity	12
2.6 The Programmable Block Construction	13
2.6.1 Overview	13
2.6.2 Architecture of the Blocks	14
2.6.2.1 Growth Blocks	14
2.6.2.2 Seed Block	17
2.6.3 The Unpacking Process	17
2.6.4 Programming Blocks and the Value of the Scaling Factor c	21
2.6.5 Uniqueness of the Terminal Assembly	21
2.7 Generalizations of Shape Complexity	22
2.7.1 Optimizing the Main Result (Section 2.5)	22
2.7.2 Strength Functions	23
2.7.3 Wang Tiling vs Self-Assembly of Shapes	23
2.7.4 Sets of Shapes	25
2.7.5 Scale Complexity of Shape Functions	26
2.7.6 Other Uses of Programmable Growth	27
2.8 Appendix	28
2.8.1 Local Determinism Guarantees Unique Production: Proof of Theorem 2.3.1	28
2.8.2 Scale-Equivalence and “ \cong ” are Equivalence Relations	28
Bibliography	29

3	Complexity of Compact Proofreading for Self-Assembled Patterns	31
3.1	Abstract	31
3.2	Introduction	31
3.2.1	The Abstract Tile Assembly Model	33
3.2.2	The Kinetic Tile Assembly Model and Errors	33
3.2.3	Quarter-Plane Patterns	34
3.3	Making Self-Assembly Robust	35
3.4	Compact Proofreading Schemes for Simple Patterns	37
3.5	A Lower Bound	39
3.6	Appendix	41
3.6.1	Extension of Chen and Goel's Theorem to Infinite Seed Boundary Assemblies	41
3.6.2	An Overlay Proofreading Scheme	41
	Bibliography	43
4	Combining Self-Healing and Proofreading in Self-Assembly	45
4.1	Abstract	45
4.2	Introduction	45
4.3	Modeling Errors	48
4.3.1	Erroneous Tile Additions During Growth	48
4.3.2	Wholesale Removal of Tiles	49
4.4	Self-Healing Proofreading Construction	49
4.5	Extensions	54
4.5.1	Random Walks in the $r_2 \neq f$ Regime	54
4.5.2	Preventing Spurious Nucleation	55
	Bibliography	56
5	The Computational Power of Benenson Automata	58
5.1	Abstract	58
5.2	Introduction	58
5.3	Formalization of Benenson Automata	61
5.4	Characterizing the Computational Power of Benenson Automata	62
5.5	Simulating Branching Programs and Circuits	63
5.5.1	General Branching Programs	63
5.5.2	Fixed-Width Branching Programs	64
5.5.3	Permutation Branching Programs	66
5.5.4	Simulating Circuits	68
5.5.5	Achieving 1-Encoded Automata	68
5.6	Shallow Circuits to Simulate Benenson automata	69
5.7	Discussion	70
	Bibliography	71
6	Computation with Finite Stochastic Chemical Reaction Networks	72
6.1	Abstract	72
6.2	Introduction	72
6.3	Stochastic Model of Chemical Kinetics	73
6.4	Time/Space-Bounded Algorithms	74
6.5	Unbounded Algorithms	79
6.6	Discussion	81
6.7	Appendix	81
6.7.1	Clock Analysis	81
6.7.2	Time/Space-Bounded RM Simulation	82
6.7.3	Time/Space-Bounded CTM Simulation	83
6.7.4	Unbounded RM Simulation	84
6.7.5	Unbounded CTM Simulation	84

6.7.6	Decidability of Reachability	85
	Bibliography	85
7	Robust Stochastic Chemical Reaction Networks	88
7.1	Abstract	88
7.2	Introduction	88
7.3	Model and Definitions	90
7.4	Robustness Examples	91
7.5	Bounded Tau-Leaping	93
7.5.1	The Algorithm	93
7.5.2	Upper Bound on the Number of Leaps	95
7.6	On the Computational Complexity of the Prediction Problem for Robust SSA Processes	96
7.7	Discussion	98
7.8	Appendix	98
7.8.1	Proof of Theorem 7.5.1: Upper Bound on the Number of Leaps	98
7.8.2	Proving Robustness by Monotonicity	102
7.8.3	Robust Embedding of a TM in an SCRN	103
7.8.4	Proof of Theorem 7.6.1: Lower Bound on the Computational Complexity of the Prediction Problem	105
7.8.5	On Implementing BTL on a Randomized TM	106
	Bibliography	106
8	Enzyme-Free Nucleic Acid Logic Circuits	109
8.1	Abstract	109
8.2	Introduction	109
8.3	Gate Construction and Verification	110
8.4	Circuit Construction	110
8.5	Conclusion	114
	Bibliography	114
9	DNA as a Universal Substrate for Chemical Kinetics	116
9.1	Abstract	116
9.2	Introduction	116
9.3	Cascades of Strand Displacement Reactions	117
9.4	Arbitrary Unimolecular Reactions	119
9.5	Arbitrary Bimolecular Reactions	120
9.6	Systematic Construction	122
9.7	Example	123
9.8	Conclusion	125
	Bibliography	125

Acknowledgements

I thank Erik Winfree for being an ideal advisor, who inspired me, challenged me, and let me have the freedom to go on my own. He is an exceptional scientist and a good human being. Amazingly, everything I found interesting, he found interesting too. The network of relations that is my brain is in profound debt to Matthew Cook for demonstrations of thinking clearly. Erik and Matt have been my role models and I hope will continue to be my friends.

I thank my collaborator and friend Georg Seelig, who taught me physics. Paul W.K. Rothmund was my famed office-mate who taught me to fly RC airplanes and how to throw a boomerang like a man. I thank Ho-Lin Chen, Rebecca Schulman, Nadine Dabby, Joseph Schaeffer, Dave Zhang, Rizal Hariadi, Jongmin Kim, Peng Yin, Lulu Qian, and all the members of the Winfree group, past and present, for making the place as friendly and engaging as it was, and for wonderful, irreplaceable discussions.

I thank Yaser Abu-Mostafa for a wonderful rotation experience, Shuki Bruck for inviting me to his stimulating group meetings, Chris Umans for teaching a fascinating complexity theory course, and John Doyle for providing positive feedback on my thesis work.

I thank the ARCS Foundation, NSF, NIH, and the Caltech CNS program for funding.

My mom and dad, and my grandma, have been my foundation. Their sacrifices for my sake are immeasurable. I cannot be grateful enough to have been born into my family, to have had my childhood, and to have them live only a drive away now. I thank my wife's family for their support from the very first time we met. To thank my wife Esther for everything she has meant to me demands more than acknowledgment, and loving her dearly is only the first step. This thesis is dedicated to our son Elie.

Abstract

Many endeavors of molecular-level engineering either rely on biological material such as nucleic acids and restriction enzymes, or are inspired by biological processes such as self-assembly or cellular regulatory networks. This thesis develops theories on three such topics: self-assembled nanostructures, molecular automata, and chemical reaction networks. The abstractions and underlying methods of the theories presented herein are based on computer science and include Turing machines and circuits. Toward engineering self-assembled nanostructures, we create a theory of scale-free shapes in which the complexity of their self-assembly is connected to the shapes' descriptonal complexity. Further, we study patterns in terms of whether they can be self-assembled robustly without an increase in scale to accommodate redundancy. We also describe a new method of ensuring resilience to more types of error simultaneously. Toward creating molecular automata we study the computational power of a restriction enzyme-based automaton. Toward designing chemical reaction networks, we develop a technique of storing and processing information in molecular counts, which is capable of achieving Turing universal computation. We also study the computational complexity of simulating stochastic chemical reaction networks and formally connect robustness and simulation efficiency. Lastly, we describe nucleic acid implementations of Boolean logic circuits and arbitrary mass-action kinetics. The three areas of this thesis are promising realizations of molecular-level engineering, and the theories presented here inform the range of possibility or delineate inherent difficulties in these areas.

Chapter 1

Introduction

A note on the use of “we”: As a matter of style and to emphasize that most of the work presented here was done in collaboration with others, “we” is used throughout the thesis, except in places where the personal voice of the author is expected.

1.1 The Promise of Bionanotechnology

Can engineers manipulate molecules with the same ease that they can now construct macrolevel objects? It is widely expected that the enabling technologies will become practical in this century [8]. Since the complexity of living things is as yet unparalleled by the engineering feats of man, life may be the ultimate example and proof of principle. Indeed one may consider the supreme promise of molecular engineering is to give science all the tools available to life. Part of the challenge is also to interact with existing biological systems and modify their function — for example to cure disease [4]. Bionanotechnology is one name of the emerging field of bioinspired molecular-level engineering. It is innately multidisciplinary and interrelates with molecular biology, synthetic biology, nanotechnology, chemical engineering, computer science, as well as many other areas. While its boundaries are fuzzy, we can define it by the premise that biology has important contributions for engineering with molecules.

Already one of the key contributions of bionanotechnology is the adaptation of nucleic acids for engineering tasks. Rather than solely acting as the carrier of genetic information, DNA has been used to construct nanoscale structures and even mechanical nanomachines [25]. Two main factors make nucleic acids the design substrate of choice. First, rational design with nucleic acids is tremendously simplified by the specificity and predictability of Watson-Crick base pairing. In general the RNA/DNA 3-D folding problem, akin to the protein folding problem, remains beyond the grasp of current computational methods. However, research suggests that keeping track of secondary structure only (which bases bind to each other) is often sufficient for understanding interactions between nucleic acids and for predicting the 3-D structure of rationally designed complexes. Secondary structure prediction and design is computationally reasonable [7]. This is in stark contrast to designing proteins, which requires modeling far more complex 3-space interactions between constituent amino acids. The second reason for the success of nucleic acids in bionanotechnology is that short DNAs and RNAs with arbitrary sequences are cheap to synthesize. Companies, such as IDT, specialize in the synthesis of short (< 100 bases) DNA or RNA oligos. Current costs are less than a dollar per base for reasonable experimental quantities, with prices steadily dropping. Longer strands can be manufactured, replicated, and otherwise manipulated using methods adopted from biotechnology. For these reasons DNA is used as the chemical substrate for most of the work in this thesis.

In this thesis I shall focus on three topics falling within the realm of bionanotechnology: the engineering of self-assembled nanostructures, the design of molecular machines for biocompatible computation, and the analysis and design of networks of chemical reactions in dilute solutions.

Advances in self-assembly have been arguably the most prominent developments in bionanotechnology. While we can exercise atomic-level engineering precision in the sense of positioning individual atoms and molecules with an atomic force microscope or scanning tunnel microscope on a surface [16, 9], this manipula-

tion requires vast investments of resources and time. Significantly each object has to be fabricated separately. In contrast, nature shapes the intrinsic properties of molecules to direct their self-organization. The fabrication of such object occurs with massive parallelism (e.g., the classic example is virus capsid self-assembly). Engineering self-assembly processes is a potentially promising way to adopt nature's construction methods. By positioning complementary domains appropriately on one or more DNA molecules, a number of 3-D shapes have been constructed, including cubes and octahedra [6, 26]. These objects assemble in bulk upon the introduction of their components into solution. One of the more recent advances has been Rothemund's DNA "origami." By designing short strands of DNA to effectively staple a long strand into a rastered form, he was able to weave smiley faces and other 2D shapes [21].

Designing shapes via above methods requires making every part of a shape out of different components, up to symmetry. In another important example, Winfree adopted Wang tiling theory [28] to control crystallization [29, 30, 22] (see below). Then by designing the individual components (tiles) to have the right local interactions, the global form and pattern of the constructed assembly can be controlled. Each type of tile may be reused multiple times in the assembly process, akin to a loop in a computer program; consequently the resulting assembly process is often called algorithmic.

Another area of significant progress in bionanotechnology has been in molecular computation. In his ground-breaking experiments, Adleman showed that by encoding mathematical information in DNA sequences it is possible to solve difficult combinatorial problems [1, 19]. The original hope of using the massive parallelism of DNA computation to exceed the power of digital computers has not realized. However, since DNA is a biocompatible computational medium it may be possible to use DNA computers to imbue cells with synthetic information processing capability which may augment or modify cellular regulatory networks. For example, Benenson et al made an automaton *in vitro* consisting of a restriction enzyme and DNA molecules which tests whether particular RNA molecules are present or not, and releases an output DNA molecule only if the combinatorial condition is met [3]. The computation consists of the enzyme sequentially cutting a DNA molecule in a manner indirectly determined by the input mRNA molecules present in solution. Benenson et al. hope that such a design may form the basis of a "smart drug," capable of detecting specific mRNA transcripts that are indicative of cancer or other diseases, and outputting a "therapeutic" ssDNA. Automata based on other kinds of protein-DNA interactions have also been constructed [2].

Much of natural information processing within a cell seems to be in regulatory networks which must respond appropriately to changes in the environment. Some of the best understood regulatory networks involve transcription-level activation and inhibition. Some complex regulatory networks have been engineered and incorporated into cells. For example, Elowitz et al. have created a synthetic oscillator which periodically changes the color of a cell based on transcriptional regulators [10]. DNA/RNA transcriptional switches have been designed to implement bistable dynamic behavior and oscillation [18, 17]. Engineered networks based on transcriptional regulation can implement Boolean circuits (AND, OR, NOT logic gates) *in vivo* and in cell-free extracts [15, 20]. If one ignores mechanistic details such as protein-protein and protein-DNA interactions, such regulatory networks can be simply expressed as systems of chemical reactions in well-mixed solutions. These chemical reaction network models are usually sufficient to explain the observed behavior; thus we call these efforts chemical reaction network engineering.

1.2 The Mathematics of Computer Science

Computer science has developed a set of theoretical tools and a perspective on the world that can be applied to many fields of human endeavor outside of programming computers. It is an unfortunate burden of computer science to be confused with IT (information technology) by the layman; it is proper to classify computer science as a branch of applied mathematics.

The three contributions of computer science most relevant for this thesis are as follows: First is the formalization of the notion of computation and the recognition of computation as a universal phenomenon. Computation is now found everywhere in nature and in mathematical abstractions, and informs many branches of science from physics to biology. The ubiquity of computation has been popularized for the wider audience by Wolfram [33]. The second contribution is asymptotic analysis. Although asymptotic analysis is not limited to computer science, it has taken off there as a consequence of making the theory of computation time insensitive to implementation details. Asymptotic analysis of functions is helpful for deriving fundamental

distinctions without getting lost in the details. For example since an exponential function is fundamentally larger than a polynomial function, it is not necessary to trace out the polynomial exponent in the derivations. In this thesis asymptotic analysis is used throughout, and computation, be it with Turing machines or circuits, makes up the heart of most of the theories presented. Lastly, computer science brings a degree of mathematical rigor to otherwise fuzzy areas of science. When objects are formally defined (or at least formally definable), it becomes possible to communicate with less fear of being misunderstood. Often the difficult part of research is isolating what is important from what is not. Having to formally define one's model forces the fixation on the essential kernel of the problem.

1.3 The Criteria for Success

The work in this thesis is largely theoretical. I propose two criteria for success of the theories developed here. First I feel the work must develop an elegant theory, although, of course, elegance is in the eye of the beholder and it is difficult to describe what makes something possess it. Second, the work must be meaningful outside of the artificial world of mathematical abstractions, and must have a take-home message for molecular-level engineering. One form of the message may be expanding the horizons of what was previously thought possible. Another may be in showing that something is inherently impossible, saving researchers time and effort. Yet another message may be a way of systematizing messy existing knowledge. I would like to think that each of these is achieved by one or another theory presented here.

A major inspiration for the manner of this work has been Winfree's tile assembly model [29], which the next three chapters directly address (see above). In Winfree's ground-breaking work a computer-theoretic abstraction called Wang tiling is realized in chemistry. The model is simple and elegant, and at the same time leads to a conclusion with significant practical implications: that, per design instructions, simple crystallization processes can be programmed to assemble arbitrarily complex structures. Recent work even suggests how this type of crystal growth may possess certain properties of life, informing the origin of life debate [24].

1.4 The Contributions

1.4.1 Tile Self-Assembly

Chapters 2, 3, and 4 concern molecular engineering through programming the process of crystallization. The underlying model is that of Winfree's tile assembly, which formalizes the two-dimensional self-assembly of square units (tiles) using a physically plausible abstraction of crystal growth [29, 30]. A new tile becomes incorporated into the growing lattice if it binds the preexisting structure strongly enough, with the strength of the bond between one tile and another being a function of the "glues" (bond types) on the abutting sides. The usual molecular implementation uses DNA hybridization to define the glues, with complementary sequences matching. At some locations in order to attach to the existing assembly a tile must bind two neighboring tiles cooperatively. Winfree showed that this cooperativity allows for surprisingly rich behavior including Turing universality.

One common goal of the assembly process in theory and experiment is to assemble a certain shape. The natural question to ask is which shapes are easy to build through self-assembly and which are hard. Since each type of tile (defined by its glues) can be mass produced, but designing each new tile type requires significant effort, the number of tile types is the natural complexity measure [23]. Previous works have studied only the construction of certain specific shapes as examples. In Chapter 2 we show that the natural formulation of the complexity problem is the construction of scale-free shapes where the scale of the shape does not matter, only the form. Somewhat paralleling the way that invariance with respect to the coordinate system turns topology into an elegant theory, our scale-free formalization allows for a number of compelling results. We prove that the minimal number of tile types required is a function of the Kolmogorov complexity of the shape. Thus the class of shapes that are easy to build are mathematically well defined. Our result also shows that the very simple process of crystallization is nonetheless powerful enough to turn any description of shape into the actual shape, fulfilling the role of von Neumann's universal constructor [27].

In Chapters 3 and 4 we turn to the problem of errors in tile self-assembly. A relatively high error rate during assembly has been one of the main setbacks in practical implementation of the theory. As opposed

to the shape of the assembly as in the previous chapter, here we consider the pattern produced by assembly: when tiles are marked with different colors (which may represent different functional domains, etc.) the assembly produces a labeled pattern [30]. Winfree and Bekbolatov first proposed to introduce redundancy into the assembly to decrease the error rate akin to a repetition code [32]. A single tile in the original error-prone tile system is replaced with a block of tiles, each carrying the information of the original tile redundantly. In Chapter 3 we ask whether the redundancy can be encoded without increasing the scale of the construction. Since scale is of direct concern here, in some ways this chapter takes the opposite approach of the theory developed in Chapter 2. Nonetheless, in practice minimizing the scale of the assembly is an important goal for certain molecular fabrication tasks. The main result of this chapter is the argument that patterns can be distinguished into two classes when scale-up is undesired: it is easy to make certain patterns robust to errors by embedding redundancy, while for other patterns this is difficult without an exponential increase in the number of tile types.

There is an additional type of error that has been studied: damage to the completed structure [31]. Chunks of an assembly may be physically ripped off by external mechanical forces, such as shear due to fluid flow during sample handling. It is desired that the constructed structure should be self-healing in the sense of being able to repair itself, or at the very least to not regrow incorrectly. Erroneous addition of tiles and assembly damage have previously been studied separately; however, existing constructions for making a tile system robust to the two types of error could not be directly combined. In Chapter 4 we develop a new method to make a pattern tile system simultaneously robust to both types of error.

1.4.2 Restriction Enzyme Automata

We next consider engineering chemical computers capable of taking certain chemical species as inputs and producing certain other species as output. One long-term goal of this type of engineering is to be able to create biocompatible computers capable of interacting with intracellular processes and modifying or augmenting the behavior of a cell in a complex way. In Chapter 5 we study the computational power of a molecular automaton recently proposed and implemented *in vitro* by Benenson et al. [3]. In this scheme, a long DNA state molecule is cut repeatedly by a restriction enzyme in a manner dependent upon the presence of particular short DNA rule molecules.

How much computation can Benenson automata perform? It was known that a single automaton can compute the conjunction of inputs (and negated inputs), yet, for example, it was not clear whether a single Benenson automaton can compute a disjunction of conjunctions. Surprisingly, we show that a single automaton is more powerful than expected and can compute arbitrary Boolean functions. We also determine the class of functions that a Benenson automaton can compute efficiently: exactly those functions computable by log-depth circuits.

1.4.3 Chemical Reaction Networks

A central feature of engineering in solution-based chemistry is the difficulty in controlling the targeting and time of interactions: any two molecules may meet and the meeting will occur at an unknown time. In biology as well as computer engineering, the resolution of this difficulty often requires space and geometry: information is stored in a polymer such as DNA, on a tape, in spatial compartments, or in different locations on a chip. For example the Benenson automaton relies on the ordered arrangement of information within a polymer which can then be sequentially extracted by cutting operations. Another example is the classic model of chemical Turing universal computation: Bennett's DNA-inspired polymer automaton [5]. Yet clearly some of the computation occurring within a cell is in the complex interactions of concentrations of some species and does not rely on space and geometry for information processing.

Consequently, we ask in Chapter 6: what intrinsic limits are placed on computation solely by the nature of well-mixed chemistry? To answer this, we develop a theoretical way of storing and manipulating information in the exact molecular counts of some molecules. In our construction, the tape of a Turing machine is mapped to the molecular count of a "memory species," and this molecular count is manipulated in a way that corresponds to a Turing machine computation. Our results say that, surprisingly, geometry is not necessary for effective computation and well-mixed chemistry is Turing universal. We show that well-mixed chemistry

can perform an a priori unbounded number of computational steps with an arbitrarily small (non-zero) cumulative error probability. However, geometry seems necessary for error-free Turing universal computation: we show that error-free Turing universal computation with chemical kinetics is impossible. Finally we show that although information is stored in unary molecular counts, requiring volumes that potentially grow exponentially in the number of Turing machine steps, the information can be manipulated quickly. The time for our construction to simulate a run of a Turing machine grows only polynomially in the number of executed steps.

The behavior of coupled chemical reactions over time is at the foundation of chemistry. Traditionally such systems are modeled with mass-action kinetics. However, numerous works have shown that in molecular biology, it is often inappropriate to use mass-action laws to simulate certain intracellular pathways. Indeed, when key molecular components are present in small molecular counts, as they often are, stochastic effects can be physiologically significant. Our construction of Chapter 6 relied on having a single molecule of certain species. Such systems effectively behave as continuous-time Markov processes, as opposed to systems described by the deterministic and continuous variation of concentration over time. Their simulation is deemed to be significantly more difficult than large-volume systems susceptible to the mass-action approximation [14]. Yet simulation of these systems is essential for understanding cellular pathways and for intracellular molecular engineering [14].

In Chapter 7 we consider the computational complexity of simulating a given stochastic chemical reaction network. Gillespie's stochastic simulation algorithm (SSA) can be used to model stochastic chemical systems [12]. However, SSA is prohibitively slow for many applications, especially when there are certain species whose molecular counts are large in addition to those species whose molecular counts are small. Recently tau-leaping algorithms have been developed that can be significantly faster than SSA, yet at the cost of certain systematic errors entering the simulation [13]. Despite the widespread use of stochastic simulation algorithms for chemical reactions, the study of their computational complexity is nascent. In fact, the speeds of different algorithms are generally compared only through specific numerical examples.

We attempt to develop a theory of the asymptotic computational complexity of a class of stochastic chemical reactions networks. First we formalize the previously implicit condition on the reaction network that guarantees the accuracy of tau-leaping algorithms: robustness to perturbations in reaction propensities. Then by using the tools of computational complexity theory, we develop closely matching asymptotic upper and lower bounds on the computational time that may be required for predicting the behavior of these systems. In the process, we define a new stochastic simulation algorithm we call bounded tau-leaping. The lower bound on the required computation time is based on the ability of the chemical system itself to perform computation in a manner akin to our construction of Chapter 6.

The major reason mass-action chemical systems are easier to simulate is because the computation time is not dependent on the total molecular count but just on concentration (molecular count per volume). Since the concentration is always bounded (the solution must be dilute enough to remain well-mixed and not a solid) the computational complexity of simulating mass action systems does not scale with the size of the system. In stark contrast, the computation time of SSA may scale linearly with the total molecular count even for bounded concentrations. Surprisingly, we show that, assuming bounded concentrations, for robust stochastic chemical systems the required computation time is asymptotically essentially invariant with molecular count. In this sense simulation of robust stochastic chemical reaction networks approaches the speed of mass action systems.

In the last two chapters, we consider the design and implementation of chemical reaction networks in practice. We restrict ourselves to the mass-action regime because single-molecule experiments were beyond the capability of the available equipment in our laboratory.

As nucleic acid interactions are easily programmable through sequence design, DNA seemed a natural choice for the chemical substrate with which to design reaction networks. Indeed, in Chapter 8 we show how DNA hybridization alone can be used to design the simplest kind of reaction networks: feedforward circuits. By maintaining digital abstraction over multiple layers we show that such circuits can operate reliably in vitro. To our knowledge our work is the first to experimentally demonstrate reliable circuits based solely on hybridization capable of multi-layer cascading. Since biological nucleic acids such as microRNAs can serve as inputs, several intriguing applications in biotechnology and bioengineering may be possible. A major contribution of this work is the mechanism of attaining modularity using toehold-mediated branch migration [34]. Hiding the toehold of an output strand keeps it inactive until intended release.

The last chapter (Chapter 9) asks: How can we implement arbitrary systems of coupled chemical reactions

with realistic chemistry? It is well known that “chemical” ordinary differential equations are capable of very complex dynamic behavior, including oscillation with limit cycles and chaos [11]. Some experimental chemical systems have even been found to behave in this manner. Yet there was not a systematic way to implement an arbitrary system of chemical ODEs in actual chemistry. Here, using the method of obtaining modularity from Chapter 8, we theoretically construct reaction cascades with arbitrary unimolecular and bimolecular kinetics. Individual reactions can be coupled in arbitrary ways such that reactants can participate in multiple reactions simultaneously, reproducing the desired dynamic behavior.

Bibliography

- [1] L.M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, 1994.
- [2] R. Bar-Ziv, T. Tlusty, and A. Libchaber. Protein–DNA computation by stochastic assembly cascade. *Proceedings of the National Academy of Sciences*, 99(18):11589, 2002.
- [3] Y. Benenson, B. Gil, U. Ben-dor, R. Adar, and E. Shapiro. An autonomous molecular computer for logical control of gene expression. *Nature*, 429:423–429, 2004.
- [4] S.A. Benner and A.M. Sismour. Synthetic biology. *Nature Review Genetics*, 6(7):533–543, 2005.
- [5] C. H. Bennett. The thermodynamics of computation — a review. *International Journal of Theoretical Physics*, 21(12):905–939, 1982.
- [6] J. Chen and N.C. Seeman. Synthesis from DNA of a molecule with the connectivity of a cube. *Nature*, 350(6319):631–633, 1991.
- [7] R.M. Dirks, J.S. Bois, J.M. Schaeffer, E. Winfree, and N.A. Pierce. Thermodynamic analysis of interacting nucleic acid strands. *SIAM Review*, 49(1):65–88, 2007.
- [8] K.E. Drexler. *Nanosystems: molecular machinery, manufacturing, and computation*. John Wiley & Sons, Inc., New York, NY, USA, 1992.
- [9] D.M. Eigler and E.K. Schweizer. Positioning single atoms with a scanning tunnelling microscope. *Nature*, 344(6266):524–526, 1990.
- [10] M.B. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–8, 2000.
- [11] I. R. Epstein and J.A. Pojman. *An Introduction to Nonlinear Chemical Dynamics: Oscillations, Waves, Patterns, and Chaos*. Oxford University Press, 1998.
- [12] D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81:2340–2361, 1977.
- [13] D.T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *Journal of Chemical Physics*, 115, 2001.
- [14] D.T. Gillespie. Stochastic simulation of chemical kinetics. *Annual Review of Physical Chemistry*, 58:35–55, 2007.
- [15] J. Hasty, D. McMillen, and J.J. Collins. Engineered gene circuits. *Nature*, 420(6912):224–230, 2002.
- [16] T. Junno, K. Deppert, L. Montelius, and L. Samuelson. Controlled manipulation of nanoparticles with an atomic force microscope. *Applied Physics Letters*, 66:3627, 1995.
- [17] J. Kim. *In vitro synthetic transcriptional networks*. PhD thesis, California Institute of Technology, 2006.
- [18] J. Kim, K.S. White, and E. Winfree. Construction of an in vitro bistable circuit from synthetic transcriptional switches. *Molecular Systems Biology*, 2(68), 2006.

- [19] R.J. Lipton. DNA solution of hard computational problems. *Science*, 268(5210):542–545, 1995.
- [20] V. Noireaux, R. Bar-Ziv, and A. Libchaber. Principles of cell-free genetic circuit assembly. *Proceedings of the National Academy of Sciences*, 100(22):12672–12677, 2003.
- [21] P.W.K. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006.
- [22] P.W.K. Rothemund, N. Papadakis, and E. Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology*, 2(12):e424, 2004.
- [23] P.W.K. Rothemund and E. Winfree. The program-size complexity of self-assembled squares (extended abstract). In *ACM Symposium on Theory of Computing*, pages 459–468, 2000.
- [24] R. Schulman and E. Winfree. How crystals that sense and respond to their environments could evolve. *Natural Computing (online)*, 2007.
- [25] N.C. Seeman. DNA in a material world. *Nature*, 421(6921):427–31, 2003.
- [26] W.M. Shih, J.D. Quispe, and G.F. Joyce. A 1.7-kilobase single-stranded DNA that folds into a nanoscale octahedron. *Nature*, 427(6975):618–621, 2004.
- [27] J. von Neumann. *The Theory of Self Reproducing Automata*. University of Illinois Press, 1966.
- [28] H. Wang. Proving theorems by pattern recognition I. *Communications of the ACM*, 3(4):220–234, 1960.
- [29] E. Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, 1998.
- [30] E. Winfree. Simulations of computing by self-assembly. *DIMACS: DNA-Based Computers*, 1998.
- [31] E. Winfree. Self-healing tile sets. In Junghuei Chen, Natasha Jonoska, and Grzegorz Rozenberg, editors, *Nanotechnology: Science and Computation*, pages 55–78, Berlin Heidelberg, 2006. Springer-Verlag.
- [32] E. Winfree and R. Bekbolatov. Proofreading tile sets: Error-correction for algorithmic self-assembly. In *DNA Computing 9*, pages 126–144, 2004.
- [33] S. Wolfram. *A New Kind of Science*. Wolfram Media, 2002.
- [34] B. Yurke and A.P. Mills. Using DNA to power nanostructures. *Genetic Programming and Evolvable Machines*, 4(2):111–122, 2003.

Chapter 2

Complexity of Self-Assembled Shapes

This chapter was published as: David Soloveichik and Erik Winfree, “Complexity of Self-Assembled Shapes,” *SIAM Journal on Computing* 36 (6): 1544–1569, 2007.

2.1 Abstract

The connection between self-assembly and computation suggests that a shape can be considered the output of a self-assembly “program,” a set of tiles that fit together to create a shape. It seems plausible that the size of the smallest self-assembly program that builds a shape and the shape’s descriptonal (Kolmogorov) complexity should be related. We show that when using a notion of a shape that is independent of scale, this is indeed so: in the Tile Assembly Model, the minimal number of distinct tile types necessary to self-assemble a shape, at some scale, can be bounded both above and below in terms of the shape’s Kolmogorov complexity. As part of the proof, we develop a universal constructor for this model of self-assembly that can execute an arbitrary Turing machine program specifying how to grow a shape. Our result implies, somewhat counter-intuitively, that self-assembly of a scaled-up version of a shape often requires fewer tile types. Furthermore, the independence of scale in self-assembly theory appears to play the same crucial role as the independence of running time in the theory of computability. This leads to an elegant formulation of languages of shapes generated by self-assembly. Considering functions from bit strings to shapes, we show that the running-time complexity, with respect to Turing machines, is polynomially equivalent to the scale complexity of the same function implemented via self-assembly by a finite set of tile types. Our results also hold for shapes defined by Wang tiling — where there is no sense of a self-assembly process — except that here time complexity must be measured with respect to non-deterministic Turing machines.

2.2 Introduction

Self-assembly is the process by which an organized structure can spontaneously form from simple parts. The Tile Assembly Model [21, 22], based on Wang tiling [20], formalizes the two-dimensional self-assembly of square units called “tiles” using a physically plausible abstraction of crystal growth. In this model, a new tile can adsorb to a growing complex if it binds strongly enough. Each of the four sides of a tile has an associated bond type that interacts with a certain strength with matching sides of other tiles. The process of self-assembly is initiated by a single seed tile and proceeds via the sequential addition of new tiles. Confirming the physical plausibility and relevance of the abstraction, simple self-assembling systems of tiles have been built out of certain types of DNA molecules [23, 15, 14, 12, 10]. The possibility of using self-assembly for nanofabrication of complex components such as circuits has been suggested as a promising application [6].

The view that the “shape” of a self-assembled complex can be considered the output of a computational process [2] has inspired recent interest [11, 1, 3, 9, 4]. While it was shown through specific examples that self-assembly can be used to construct interesting shapes and patterns, it was not known in general which shapes could be self-assembled from a small number of tile types. Understanding the complexity of shapes is facilitated by an appropriate definition of shape. In our model, a tile system generates a particular shape

if it produces any scaled version of that shape (Section 2.4). This definition may be thought to formalize the idea that a structure can be made up of arbitrarily small pieces, but more importantly this leads to an elegant theory that is impossible to achieve without ignoring scale. Computationally, it is analogous to disregarding computation time and is thus more appropriate as a notion of output of a *universal* computation process.* Using this definition of shape, we show (Section 2.5) that for any shape \tilde{S} , if $K_{sa}(\tilde{S})$ is the minimal number of distinct tile types necessary to self-assemble it, then $K_{sa}(\tilde{S}) \log K_{sa}(\tilde{S})$ is within multiplicative and additive constants (independent of \tilde{S}) of the shape’s Kolmogorov complexity. This theorem is proved by developing a universal constructor [19] for self-assembly which uses a program that outputs a fixed size shape as a list of locations to make a scaled version of the shape (Section 2.6). This construction, together with a new proof technique for showing that a tile set produces a unique assembly (*local determinism*), might be of independent interest. Our result ties the computation of a shape and its self-assembly, and, somewhat counter-intuitively, implies that it may often require fewer tile types to self-assemble a larger instance of a shape than a smaller instance thereof. Another consequence of the theorem is that the minimal number of tile types necessary to self-assemble an arbitrary scaling of a shape is uncomputable. Answering the same question about shapes of a fixed size is computable but NP complete [1].

The tight correspondence between computation (ignoring time) and self-assembly (ignoring scale) suggests that complexity measures based on time (for computation) and on scale (for self-assembly) could also be related. To establish this result, we consider “programmable” tile sets that will grow a particular member of a family of shapes, dependent upon input information present in an initial seed assembly. We show that, as a function of the length of the input information, the number of tiles present in the shape (a measure of its scale) is polynomially related to the time required for a Turing machine to produce a representation of the same shape. Furthermore, we discuss the relationship between complexities for Wang tilings (in which the existence of a tiling rather than its creation by self-assembly is of relevance) and for self-assembly, and we show that while the Kolmogorov complexity is unchanged, the scale complexity for Wang tilings is polynomially related to the time for *non-deterministic* Turing machines. These results are presented in Section 2.7.

2.3 The Tile Assembly Model

We present a description of the Tile Assembly Model based on Rothmund and Winfree [11] and Rothmund [9]. We will be working on a $\mathbb{Z} \times \mathbb{Z}$ grid of unit square locations. The **directions** $\mathcal{D} = \{N, E, S, W\}$ are used to indicate relative positions in the grid. Formally, they are functions $\mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z} \times \mathbb{Z}$: $N(i, j) = (i, j+1)$, $E(i, j) = (i+1, j)$, $S(i, j) = (i, j-1)$, and $W(i, j) = (i-1, j)$. The inverse directions are defined naturally: $N^{-1}(i, j) = S(i, j)$, etc. Let Σ be a set of **bond types**. A **tile type** $\boxed{\tau}$ is a 4-tuple $(\sigma_N, \sigma_E, \sigma_S, \sigma_W) \in \Sigma^4$ indicating the associated bond types on the north, east, south, and west sides. Note that tile types are oriented, so a rotated version of a tile type is considered to be a different tile type. A special bond type *null* represents the lack of an interaction and the special tile type *empty* = $(null, null, null, null)$ represents an empty space. If T is a set of tile types, a **tile** is a pair $(\boxed{\tau}, (i, j)) \in T \times \mathbb{Z}^2$ indicating that location (i, j) contains the tile type $\boxed{\tau}$. Given the tile $t = (\boxed{\tau}, (i, j))$, $type(t) = \boxed{\tau}$ and $pos(t) = (i, j)$. Further, $bond_D(\boxed{\tau})$, where $D \in \mathcal{D}$, is the bond type of the respective side of $\boxed{\tau}$, and $bond_D(t) = bond_D(type(t))$. A **configuration** is a set of non-*empty* tiles, with types from T , such that there is no more than one tile in every location $(i, j) \in \mathbb{Z} \times \mathbb{Z}$. For any configuration A , we write $A(i, j)$ to indicate the tile at location (i, j) or the tile $(empty, (i, j))$ if there is no tile in A at this location.

A **strength function** $g : \Sigma \times \Sigma \rightarrow \mathbb{Z}$, where $null \in \Sigma$, defines the interactions between adjacent tiles: we say that a tile t_1 interacts with its neighbor t_2 with strength $\Gamma(t_1, t_2) = g(\sigma, \sigma')$ where σ is the bond type of tile t_1 that is adjacent to the bond type σ' of tile t_2 .[†] The *null* bond has a zero interaction strength

*The production of a shape of a fixed size cannot be considered the output of a universal computation process. Whether a universal process will output a given shape is an undecidable question, whereas this can be determined by exhaustive enumeration in the Tile Assembly Model. Thus it is clear that the connection between Kolmogorov complexity and the number of tile types we obtain in our main result (Section 2.5) cannot be achieved for fixed-scale shapes: this would violate the uncomputability of Kolmogorov complexity.

[†]More formally,

$$\Gamma(t_1, t_2) = \begin{cases} g(bond_{D^{-1}}(t_1), bond_D(t_2)) & \text{if } \exists D \in \mathcal{D} \text{ s.t. } pos(t_1) = D(pos(t_2)); \\ 0 & \text{otherwise.} \end{cases}$$

(i.e., $\forall \sigma \in \Sigma, g(\text{null}, \sigma) = 0$). We say that a strength function is **diagonal** if it is non-zero only for $g(\sigma, \sigma')$ such that $\sigma = \sigma'$. Unless otherwise noted, a tile system is assumed to have a diagonal strength function. Our constructions use diagonal strength functions with the range $\{0, 1, 2\}$. We say that a bond type σ has **strength** $g(\sigma, \sigma)$. Two tiles are **bonded** if they interact with a positive strength. For a configuration A , we use the notation $\Gamma_D^A(t) = \Gamma(t, A(D(\text{pos}(t))))$.^{*} For $L \subseteq \mathcal{D}$ we define $\Gamma_L^A(t) = \sum_{D \in L} \Gamma_D^A(t)$.

A **tile system** \mathbf{T} is a quadruple (T, t_s, g, τ) where T is a finite set of non-empty tile types, t_s is a special **seed tile**[†] with $\text{type}(t_s) \in T$, g is a strength function, and τ is the threshold parameter. Self-assembly is defined by a relation between configurations. Suppose A and B are two configurations, and t is a tile such that $A = B$ except at $\text{pos}(t)$ and $A(\text{pos}(t)) = \text{null}$ but $B(\text{pos}(t)) = t$. Then we write $A \rightarrow_{\mathbf{T}} B$ if $\Gamma_D^A(t) \geq \tau$. This means that a tile can be added to a configuration iff the sum of its interaction strengths with its neighbors reaches or exceeds τ . The relation $\rightarrow_{\mathbf{T}}^*$ is the reflexive transitive closure of $\rightarrow_{\mathbf{T}}$.

Whereas a configuration can be any arrangement of tiles (not necessarily connected), we are interested in the subclass of configurations that can result from a self-assembly process. Formally, the tile system and the relation $\rightarrow_{\mathbf{T}}^*$ define the partially ordered set of **assemblies**: $\text{Prod}(\mathbf{T}) = \{A \text{ s.t. } \{t_s\} \rightarrow_{\mathbf{T}}^* A\}$, and the set of **terminal assemblies**: $\text{Term}(\mathbf{T}) = \{A \in \text{Prod}(\mathbf{T}) \text{ and } \nexists B \neq A \text{ s.t. } A \rightarrow_{\mathbf{T}}^* B\}$. A tile system \mathbf{T} **uniquely produces** A if $\forall B \in \text{Prod}(\mathbf{T}), B \rightarrow_{\mathbf{T}}^* A$ (which implies $\text{Term}(\mathbf{T}) = \{A\}$).

An **assembly sequence** \vec{A} of \mathbf{T} is a sequence of pairs (A_n, t_n) where $A_0 = \{t_0\} = \{t_s\}$ and $A_{n-1} \rightarrow_{\mathbf{T}} A_n = A_{n-1} \cup \{t_n\}$. Here we will exclusively consider finite assembly sequences. If a finite assembly sequence \vec{A} is implicit, A indicates the last assembly in the sequence.

The tile systems used in our constructions have $\tau = 2$ with the strength function ranging over $\{0, 1, 2\}$. It is known that $\tau = 1$ systems with strength function ranging over $\{0, 1\}$ are rather limited [11, 9]. In our drawings, the bond type σ may be illustrated by a combination of shading, various graphics, and symbols. Strength-2 bond types will always contain two dots in their representation. All markings must match for two bond types to be considered identical. For example, the north bond type of the following tile has strength 2 and the others have strength-1.



The constructions in this paper do not use strength-0 bond types (other than in *empty* tiles); thus, there is no confusion between strength-1 and strength-0 bond types. Zero-strength interactions due to mismatches between adjacent tiles do occur in our constructions.

2.3.1 Guaranteeing Unique Production

When describing tile systems that produce a desired assembly, we would like an easy method for showing that this assembly is uniquely produced. While it might be easy to find an assembly sequence that leads to a particular assembly, there might be many other assembly sequences that lead elsewhere. Here we present a property of an assembly sequence that guarantees that the assembly it produces is indeed the uniquely produced assembly of the tile system.

Rothmund [9] describes the deterministic-RC property of an assembly that guarantees its unique production and that is very easy to check. However, this property is satisfied only by convex (in the sense of polyaminos) assemblies and thus cannot be directly invoked when making arbitrary shapes.[‡] A more general poly-time test for unique production was also shown by Rothmund [9], but it can be difficult to prove that a particular assembly would satisfy this test. On the other hand, the notion of locally deterministic assembly sequences introduced here is easily checkable and sufficient for the constructions in this paper.

Definition 2.3.1. For an assembly sequence \vec{A} we define the following sets of directions for $\forall i, j \in \mathbb{Z}$, letting $t = A(i, j)$:

^{*}Note that $t \neq A(\text{pos}(t))$ is a valid choice. In that case $\Gamma_D^A(t)$ tells us how t would bind if it were in A .

[†]While having a single seed tile is appropriate to the complexity discussion of the main part of this paper, it is useful to consider whole *seed assemblies* (made up of tiles not necessarily in T) when considering tile systems capable of producing multiple shapes (Section 2.7.5).

[‡]Additionally, assemblies satisfying the deterministic-RC property must have no strength-0 interactions between neighboring non-empty tiles. However, such interactions are used in our construction.

- $inputsides^{\vec{A}}(t) = \{D \in \mathcal{D} \text{ s.t. } t = t_n \text{ and } \Gamma_D^{A_n}(t_n) > 0\}$,
- $propsides^{\vec{A}}(t) = \{D \in \mathcal{D} \text{ s.t. } D^{-1} \in inputsides^{\vec{A}}(A(D(pos(t))))\}$,
- $termsides^{\vec{A}}(t) = \mathcal{D} - inputsides^{\vec{A}}(t) - propsides^{\vec{A}}(t)$.

Intuitively, *inputsides* are the sides with which the tile initially binds in the process of self-assembly; these sides determine its identity. *propsides* propagate information by being the sides to which neighboring tiles bind. *termsides* are sides that do neither. Note that by definition *empty* tiles have four *termsides*.

Definition 2.3.2. A finite assembly sequence \vec{A} of $\mathbf{T} = (T, t_s, g, \tau)$ is called **locally deterministic** if $\forall i, j \in \mathbb{Z}$, letting $t = A(i, j)$,

1. $\Gamma_{inputsides^{\vec{A}}(t)}^A(t) \leq \tau$
2. $\forall t' \text{ s.t. } type(t') \in T, pos(t') = pos(t) \text{ but } type(t') \neq type(t)$,

$$\Gamma_{\mathcal{D} - propsides^{\vec{A}}(t)}^A(t') < \tau.$$

We allow the possibility of $<$ in property (1) in order to account for the seed and *empty* tiles. Intuitively, the first property says that when a new tile binds to a growing assembly, it binds “just barely.” The second property says that nothing can grow from non-propagating sides except “as desired.” We say that \mathbf{T} is locally deterministic if there exists a locally deterministic assembly sequence for it.

It is clear that if \vec{A} is a locally deterministic assembly sequence of \mathbf{T} , then $A \in Term(\mathbf{T})$. Otherwise, the *empty* tile in the position where a new (non-empty) tile can be added to A would violate the second property. However, the existence of a locally deterministic assembly sequence leads to a much stronger conclusion:

Theorem 2.3.1. If there exists a locally deterministic assembly sequence \vec{A} of \mathbf{T} then \mathbf{T} uniquely produces A .

Proof. See Appendix 2.8.1. □

2.4 Arbitrarily Scaled Shapes and Their Complexity

In this section, we introduce the model for the output of the self-assembly process used in this paper. Let S be a finite set of locations on $\mathbb{Z} \times \mathbb{Z}$. The adjacency graph $G(S)$ is the graph on S defined by the adjacency relation where two locations are considered adjacent if they are directly north/south, or east/west of one another. We say that S is a **coordinated shape** if $G(S)$ is connected.* The **coordinated shape of assembly** A is the set $S_A = \{pos(t) \text{ s.t. } t \in A\}$. Note that S_A is a coordinated shape because A constitutes a single connected component.

For any set of locations S , and any $c \in \mathbb{N}$, we define a **c -scaling of S** as

$$S^c = \{(i, j) \text{ s.t. } (\lfloor i/c \rfloor, \lfloor j/c \rfloor) \in S\}.$$

Geometrically, this represents a “magnification” of S by a factor c . Note that a scaling of a coordinated shape is itself a coordinated shape: every node of $G(S)$ gets mapped to a c^2 -node connected subgraph of $G(S^c)$ and the relative connectivity of the subgraphs is the same as the connectivity of the nodes of $G(S)$. A parallel argument shows that if S^c is a coordinated shape, then so is S . We say that coordinated shapes S_1 and S_2 are **scale-equivalent** if $S_1^c = S_2^d$ for some $c, d \in \mathbb{N}$. Two coordinated shapes are **translation-equivalent** if they can be made identical by translation. We write $S_1 \cong S_2$ if S_1^c is translation-equivalent to S_2^d for some $c, d \in \mathbb{N}$. Scale-equivalence, translation-equivalence, and \cong are equivalence relations (Appendix 2.8.2). This defines the equivalence classes of coordinated shapes under \cong . The equivalence class containing S is denoted \tilde{S} and we refer to it as the **shape of assembly** \tilde{S} . We say that \tilde{S} is the **shape of assembly** A if $S_A \in \tilde{S}$. The

*We say “coordinated” to make explicit that a fixed coordinate system is used. We reserve the unqualified term “shape” for when we ignore scale and translation.

view of computation performed by the self-assembly process espoused here is the production of a shape as the “output” of the self-assembly process, with the understanding that the scale of the shape is irrelevant. Physically, this view may be appropriate to the extent that a physical object can be constructed from arbitrarily small pieces. However, the primary reason for this view is that there does not seem to be a comprehensive theory of complexity of coordinated shapes akin to the theory we develop here for shapes ignoring scale.

Having defined the notion of shapes, we turn to their descriptonal complexity. As usual, the Kolmogorov complexity of a binary string x with respect to a universal Turing machine U is $K_U(x) = \min \{|p| \text{ s.t. } U(p) = x\}$. (See the exposition of Li and Vitanyi [13] for an in-depth discussion of Kolmogorov complexity.) Let us fix some “standard” universal machine U . We call the Kolmogorov complexity of a coordinated shape S to be the size of the smallest program outputting it as a list of locations:^{*,†}

$$K(S) = \min \{|s| \text{ s.t. } U(s) = \langle S \rangle\}.$$

The Kolmogorov complexity of a shape \tilde{S} is:

$$K(\tilde{S}) = \min \{|s| \text{ s.t. } U(s) = \langle S \rangle \text{ for some } S \in \tilde{S}\}.$$

We define the **tile-complexity** of a coordinated shape S and shape \tilde{S} , respectively, as:

$$K_{sa}(S) = \min \left\{ n \text{ s.t. } \exists \text{ a tile system } \mathbf{T} \text{ of } n \text{ tile types that uniquely produces as-} \right. \\ \left. \text{sembly } A \text{ and } S \text{ is the coordinated shape of } A \right\}$$

$$K_{sa}(\tilde{S}) = \min \left\{ n \text{ s.t. } \exists \text{ a tile system } \mathbf{T} \text{ of } n \text{ tile types that uniquely produces as-} \right. \\ \left. \text{sembly } A \text{ and } \tilde{S} \text{ is the shape of } A \right\}.$$

2.5 Relating Tile-Complexity and Kolmogorov Complexity

The essential result of this paper is the description of the relationship between the Kolmogorov complexity of any shape and the number of tile types necessary to self-assemble it.

Theorem 2.5.1. *There exist constants a_0, b_0, a_1, b_1 such that for any shape \tilde{S} ,*

$$a_0 K(\tilde{S}) + b_0 \leq K_{sa}(\tilde{S}) \log K_{sa}(\tilde{S}) \leq a_1 K(\tilde{S}) + b_1. \quad (2.1)$$

Note that since any tile system of n tile types can be described by $O(n \log n)$ bits, the theorem implies there is a way to construct a tiling system such that asymptotically at least a constant fraction of these bits is used to “describe” the shape rather than any other aspect of the tiling system.

Proof. To see that $a_0 K(\tilde{S}) + b_0 \leq K_{sa}(\tilde{S}) \log K_{sa}(\tilde{S})$, realize that there exists a constant size program p_{sa} that, given a binary description of a tile system, simulates its self-assembly, making arbitrary choices where multiple tile additions are possible. If the self-assembly process terminates, p_{sa} outputs the coordinated shape of the terminal assembly as the binary encoding of the list of locations in it. Any tile system \mathbf{T} of n tile types with any diagonal strength function and any threshold τ can be represented[‡] by a string $d_{\mathbf{T}}$ of $4n \lceil \log 4n \rceil + 16n$ bits: For each tile type, the first of which is assumed to be the seed, specify the bond types on its four sides. There are no more than $4n$ bond types. In addition, for each tile type \mathbb{U} specify for which of the 16 subsets $L \subseteq \mathcal{D}$, $\sum_{D \in L} g(\text{bond}_D(\mathbb{U})) \geq \tau$. If \mathbf{T} is a tile system uniquely producing an assembly that has shape \tilde{S} , then $K(\tilde{S}) \leq |p_{sa} d_{\mathbf{T}}|$. The left inequality in eq. 2.1 follows with the multiplicative constant $a_0 = 1/4 - \varepsilon$ for arbitrary $\varepsilon > 0$.

^{*}Note that $K(S)$ is within an additive constant of $K_U(x)$ where x is some other effective description of S , such as a computable characteristic function or a matrix. Since our results are asymptotic, they are independent of the specific representation choice. One might also consider invoking a two-dimensional computing machine, but it is not fundamentally different for the same reason.

[†]Notation $\langle \cdot \rangle$ indicates some standard binary encoding of the object(s) in the brackets. In the case of coordinated shapes, it means an explicit binary encoding of the set of locations. Integers, tuples, or other data structures are similarly given simple explicit encodings.

[‡]Note that this representation could also be used in the case that negative bond strengths are allowed so long as the strength function is diagonal.

We prove the right inequality in eq. 2.1 by developing a construction (Section 2.6) showing how for any program s s.t. $U(s) = \langle S \rangle$, we can build a tile system \mathbf{T} of $15 \frac{|p|}{\log |p|} + b$ tile types, where b is a constant and p is a string consisting of a fixed program p_{sb} and s (i.e., $|p| = |p_{sb}| + |s|$), that uniquely produces an assembly whose shape is \tilde{S} . Program p_{sb} and constant b are both independent of S . The right inequality in eq. 2.1 follows with the multiplicative constant $a_1 = 15 + \varepsilon$ for arbitrary $\varepsilon > 0$. \square

Our result can be used to show that the tile-complexity of shapes is uncomputable:

Corollary 2.5.1. *K_{sa} of shapes is uncomputable. In other words, the following language is undecidable:*
 $\tilde{L} = \left\{ (l, n) \text{ s.t. } l = \langle S \rangle \text{ for some } S \text{ and } K_{sa}(\tilde{S}) \leq n \right\}$.

Language \tilde{L} should be contrasted with $L = \{(l, n) \text{ s.t. } l = \langle S \rangle \text{ and } K_{sa}(S) \leq n\}$ which is decidable (but hard to compute in the sense of NP-completeness [1]).

Proof. We essentially parallel the proof that Kolmogorov complexity is uncomputable. If \tilde{L} were decidable, then we could make a program that computes $K_{sa}(\tilde{S})$ and subsequently uses Theorem 2.5.1 to compute an effective lower bound for $K(\tilde{S})$. Then we can construct a program p that given n outputs some coordinated shape S (as a list of locations) such that $K(\tilde{S}) \geq n$ by enumerating shapes and testing with the lower bound, which we know must eventually exceed n . But this results in a contradiction since $p\langle n \rangle$ is a program outputting $S \in \tilde{S}$ and so $K(\tilde{S}) \leq |p| + \lceil \log n \rceil$. But for large enough n , $|p| + \lceil \log n \rceil < n$. \square

2.6 The Programmable Block Construction

2.6.1 Overview

The uniquely produced terminal assembly A of our tile system logically will consist of square “blocks” of c by c tiles. There will be one block for each location in S . Consider the coordinated shape in Fig. 2.1(a). An example assembly A is graphically represented in Fig. 2.1(b), where each square represents a block containing c^2 tiles. Self-assembly initiates in the *seed block*, which contains the seed tile, and proceeds according to the arrows illustrated between blocks. Thus if there is an arrow from one block to another, it indicates that the growth of the second block (a *growth block*) is initiated from the first. A terminated arrow indicates that the block does not initiate the self-assembly of an adjacent block in that direction — in fact, the boundary between such blocks consists of strength-0 interactions (i.e., mismatches). Fig. 2.1(c) describes our nomenclature: an arrow comes into a block on its input side, arrows exit on propagating output sides, and terminated arrows indicate terminating output sides. The seed block has four output sides, which can be either propagating or terminating. Each growth block has one input and three output sides, which are also either propagating or terminating. The overall pattern of bonding of the finished target assembly A is as follows. Tiles on terminal output sides are not bound to the tiles on the adjacent terminal output side (i.e., there is no bonding along the dotted lines in Fig. 2.8(a)), but all other neighboring tiles are bound. We will program the growth such that terminating output sides abut only other terminating output sides or *empty* tiles, and input sides exclusively abut propagating output sides and vice versa.

The input/output connections of the blocks form a spanning tree rooted at the seed block. During the progress of the self-assembly of the seed block, a computational process determines the input/output relationships of the rest of the blocks in the assembly. This information is propagated from block to block during self-assembly (along the arrows in Fig. 2.1(b)) and describes the shape of the assembly. By following the instructions each growth block receives in its input, the block decides where to start the growth of the next block and what information to pass to it in turn. The scaling factor c is set by the size of the seed block. The computation in the seed block ensures that c is large enough that there is enough space to do the necessary computation within the other blocks.

We present a general construction that represents a Turing-universal way of guiding large scale self-assembly of blocks based on an input program p . In the following section, we describe the architecture of seed and growth blocks on which arbitrary programs can be executed. In Section 2.6.3 we describe how program p can be encoded using few tile types. In Section 2.6.4 we discuss the programming of p that is required to grow the blocks in the form of a specific shape and bound the scaling factor c . In Section 2.6.5 we demonstrate that the target assembly A is *uniquely* produced.

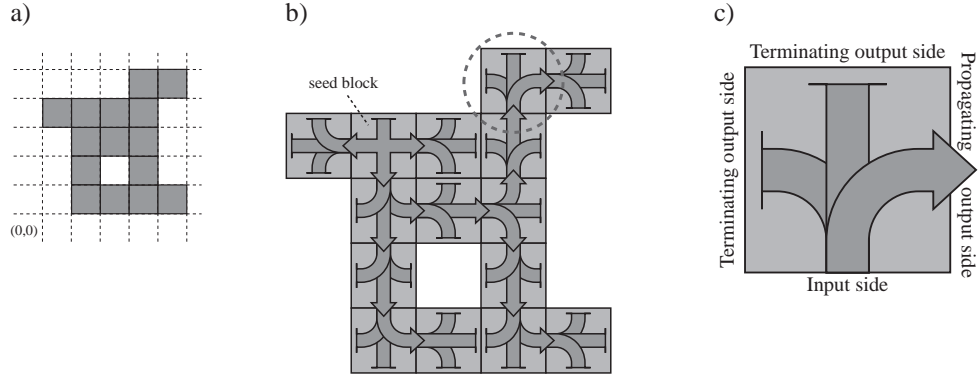


Figure 2.1: Forming a shape out of blocks: a) A coordinated shape S . b) An assembly composed of c by c blocks that grow according to transmitted instructions such that the shape of the final assembly is \tilde{S} . Arrows indicate information flow and order of assembly. (Not drawn to scale.) The seed block and the circled growth block are schematically expanded in Fig. 2.2. c) The nomenclature describing the types of block sides.

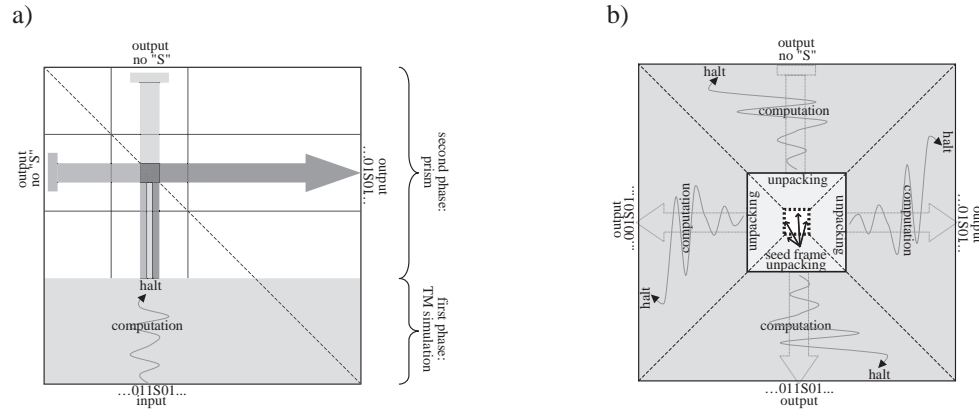


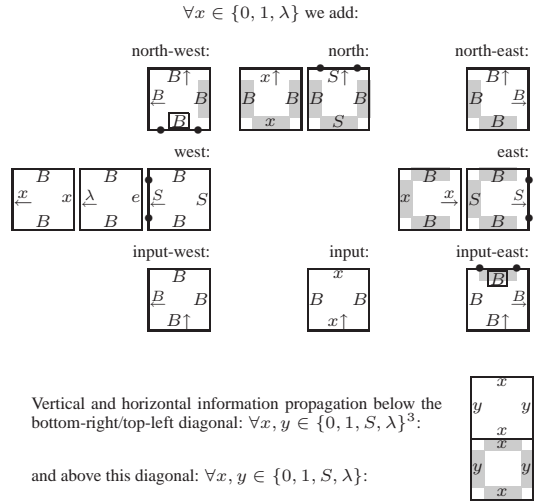
Figure 2.2: Internal structure of a growth block (a) and seed block (b)

2.6.2 Architecture of the Blocks

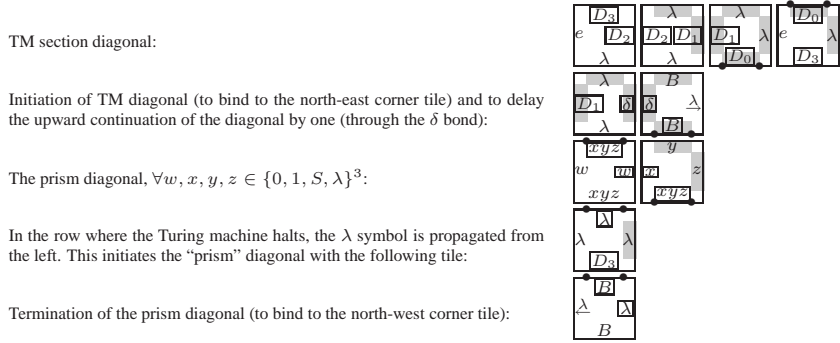
2.6.2.1 Growth Blocks

There are four types of growth blocks depending upon where the input side is, which will be labeled by \uparrow , \rightarrow , \downarrow , or \leftarrow . The internal structure of a \uparrow growth block is schematically illustrated in Fig. 2.2(a). The other three types of growth block are rotated versions of the \uparrow block. The specific tile types used for a \uparrow growth block are shown in Fig. 2.3, and a simple example is presented in Fig. 2.4. The first part is a Turing machine simulation, which is based on [18, 11]. The machine simulated is a universal Turing machine that takes its input from the propagating output side of the previous block. This TM has an output alphabet $\{0, 1, S\}^3$ and an input alphabet $\{(000), (111)\}$ on a two-way tape (with λ used as the blank symbol). The output of the simulation, as 3-tuples, is propagated until the diagonal. The diagonal propagates each member of the 3-tuples crossing it to one of the three output sides, like a prism separating the colors of the spectrum. This allows the single TM simulation to produce three separate strings targeted for the three output sides. The “ S ” symbol in the output of the TM simulation is propagated like the other symbols. However, it acts in a special way when it crosses the boundary tiles at the three output sides of the block, where it starts a new block. The output sides that receive the “ S ” symbol become propagating output sides and the output sides that do not receive it become terminating output sides. In this way, the TM simulation decides which among the three output sides will become propagating output sides, and what information they should contain, by outputting appropriate tuples. Subsequent blocks will use this information as a program, as discussed in Section 2.6.4.

a) Borders and basic info propagating tiles:



b) Tile types for the diagonal:



c) TM Simulation tile types:

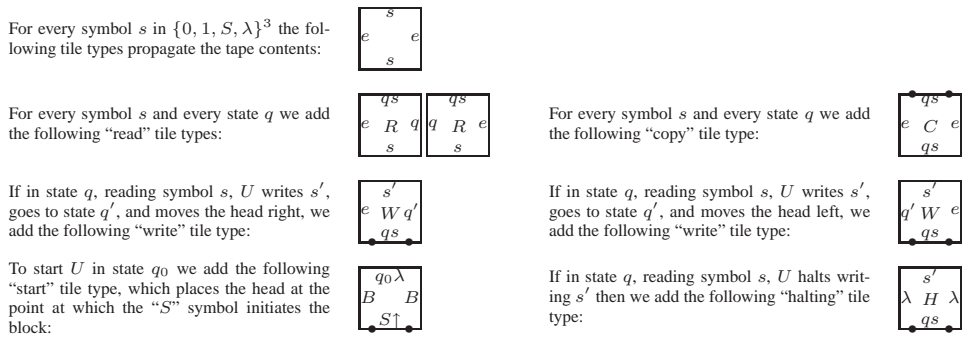


Figure 2.3: Growth block \uparrow tile types. All bond types in which a block type symbol is omitted have the block type symbol " \uparrow " to prevent inadvertent incorporation of tiles from a different block type. We assume that in bond types above, a single symbol $x \in \{0, 1, S, \lambda\}$ is the same as the tuple (xxx) . The tile types for other growth block types are formed by 90, 180, 270 degree rotations of the tile types of the \uparrow block where the block type symbols $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$ are replaced by a corresponding 90, 180, 270 degree rotation of the symbol: i.e., $\begin{matrix} B\uparrow \\ B\uparrow & B\uparrow \\ B\uparrow \end{matrix}$ (\uparrow growth block) \Rightarrow $\begin{matrix} B\rightarrow \\ B\rightarrow & B\rightarrow \\ B\rightarrow \end{matrix}$ (\rightarrow growth block). Looking at the border tile types, note that external sides of tiles on output sides of blocks have block type symbols compatible with the tiles on an input side of a block. However, tiles on output sides cannot bind to the tiles on an adjacent output side because of mismatching block type symbols.

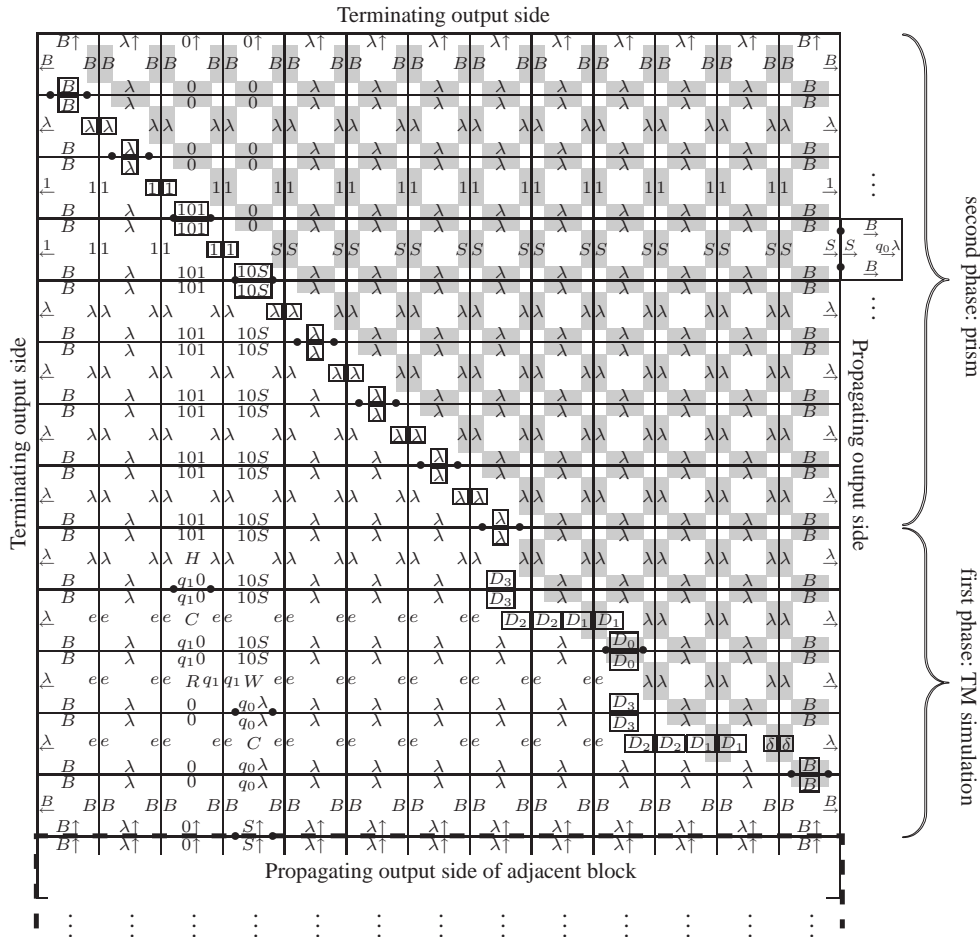


Figure 2.4: A trivial example of a \uparrow growth block. Here, the TM makes one state transition and halts. All bond types in which a block type symbol is omitted have the block type symbol “ \uparrow ”. We assume that in bond types above, a single symbol $x \in \{0, 1, S, \lambda\}$ is the same as the tuple (xxx) . The natural assembly sequence to consider is adding tiles row by row from the south side (in which a new row is started by the strength-2 bond).

2.6.2.2 Seed Block

The internal structure of the seed block is schematically shown in Fig. 2.2(b). It consists of a small square containing all the information pertaining to the shape to be built (the seed frame), a larger square in which this information is unpacked into usable form, and finally four TM simulations whose computations determine the size of the seed block and the information transmitted to the growth blocks. For simplicity we first present a construction without the unpacking process (the *simple* seed block), and then we explain the unpacking process separately and show how it can be used to create the full construction. The tile types used for the simple seed block are presented in Fig. 2.5 and an example is given in Fig. 2.6. While growth blocks contain a single TM simulation that outputs a different string to each of the three output sides, the seed block contains four *identical* TM simulations that output different strings to each of the four output sides. This is possible because the border tile types transmit information selectively: the computation in the seed block is performed using 4-tuples as the alphabet in a manner similar to the growth blocks, but on each side of the seed block only one of the elements of the 4-tuple traverses the border. As with growth blocks, if the transmitted symbol is “S,” the outside edge initiates the assembly of the adjoining block. The point of having four identical TM simulations is to ensure that the seed block is square: while a growth block uses the length of its input side to set the length of its output sides (via the diagonal), the seed block does not have any input sides. (Remember that it is the seed block that sets the size of all the blocks.)

The initiation of the Turing machine simulations in the seed block is done by tile types encoding the program p that guides the block construction. The natural approach to provide this input is using 4 rows (one for each TM) of unique tiles encoding one bit per tile, as illustrated in Figs. 2.5 and 2.6. However, this method does not result in an asymptotically optimal encoding.

2.6.3 The Unpacking Process

To encode bits much more effectively we follow Adleman et al. [3] and encode on the order of $\log n / \log \log n$ bits per tile where n is the length of the input. This representation is then unpacked into a one-bit-per-tile representation used by the TM simulation. Adleman et al.’s method requires $O(n / \log n)$ tiles to encode n bits, leading to the asymptotically optimal result of Theorem 2.5.1.

Our way of encoding information is based on Adleman et al. [3], but modified to work in a $\tau = 2$ tile system (with strength function ranging over $\{0, 1, 2\}$) and to fit our construction in its geometry. We express a length n binary string using a concatenation of $\lceil n/k \rceil$ binary substrings of length k , padding with 0s if necessary.* We choose k such that it is the least integer satisfying $\frac{n}{\log n} \leq 2^k$. Clearly, $2^k < \frac{2n}{\log n}$. See Fig. 2.7 for the tile types used in the unpacking for the north TM simulation and for a simple unpacking example (which for the sake of illustration uses $k = 4$).

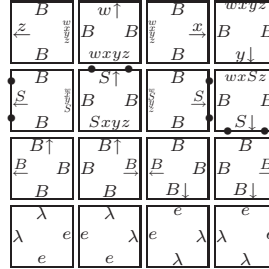
Let us consider the number of tile types used to encode and unpack the n bit input string for a single TM simulation (i.e., north). There are $2^{\lceil n/k \rceil} \leq 2^{\lceil \frac{n}{\log \frac{n}{\log n}} \rceil} = 2^{\lceil \frac{n}{\log n - \log \log n} \rceil}$ unique tile types in each seed row. This implies that there exists a constant h such that $2^{\lceil n/k \rceil} \leq \frac{3n}{\log n} + h$ for all n . We need at most $2^k + 2^{k-1} + \dots + 4 < 2^{k+1}$ “extract bit” tile types and $2^{k-1} + 2^{k-2} + \dots + 4 < 2^k$ “copy remainder” tile types. To initiate the unpacking of new substrings we need 2^k tile types. To keep on copying substrings that are not yet unpacked we need $2(2^k)$ tile types. The quantity of the other tile types is independent of n, k . Thus, in total, to unpack the n bit input string for a single TM simulation we need no more than $\frac{3n}{\log n} + h + 2^{k+1} + 2^k + 2^k + 2(2^k) \leq 15 \frac{n}{\log n} + O(1)$ tile types. Since there are 4 TM simulations in the seed block, we need $60 \frac{n}{\log n} + O(1)$ tile types to encode and unpack the n bit input string.

If the seed block requires only one propagating output side, then a reduced construction using fewer tile types can be used: only one side of the seed frame is specified, and only one direction of unpacking tiles are used. A constant number of additional tile types are used to fill out the remaining three sides of the square. These additional tile types must perform two functions. First, they must properly extend the diagonal on either side of the unpacking and TM simulation regions. In the absence of the other three unpacking and TM simulation processes, this requires adding strength-2 bonds that allow the diagonal to grow to the next layer. Second, the rest of the square must be filled in to the correct size. This can be accomplished by adding tiles that extend one diagonal to the other side of the seed frame (using the same logic as a construction in [11].)

* We can assume that our universal TM U treats trailing 0s just as λ s

a) Borders and half-diagonals:

The borders:
 $\forall w, x, y, z \in \{0, 1, \lambda\}$:

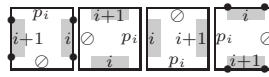


Corner tile types:

The four half-diagonals to separate the TM simulations and augment the TM tape with blanks:

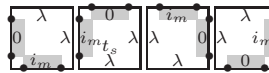
b) Seed frame for program p :

TM seed frame: for every symbol p_i :



If p_i is "U" then the corresponding bond type is strength 2, starting the TM simulation with the head positioned at that point reading λ .

Corners of the seed frame: let $i_m = |p|$:



We make the north-west corner the seed tile of our tile system.

To fill in the middle:

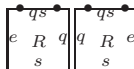


c) TM Simulation tile types (north only):

For every symbol s in $\{0, 1, S, \lambda\}^4$ the following tile types propagate the tape contents:



For every symbol s and every state q we add the following "read" tile types:



If in state q , reading symbol s , U writes s' , goes to state q' , and moves the head left, we add the following "write" tile type:



If in state q , reading symbol s , U writes s' , goes to state q' , and moves the head right, we add the following "write" tile type:



To start U in state q_0 we add the following "start" tile type, which places the head at the point at which the "S" symbol initiates the block:



If in state q , reading symbol s , U halts writing $s' = (wxyz)$ then we add the following "halting" tile type, which also starts the border:



Figure 2.5: Seed block tile types without unpacking. All bond types in which a block type symbol is omitted have the block type symbol "λ" to prevent inadvertent incorporation of tiles from a different block type. We assume that in bond types above, a single symbol $x \in \{0, 1, S, \lambda\}$ is the same as the tuple $(xxxx)$. Note that as with output sides of growth blocks, the external sides of seed block border tiles have block type symbols compatible with the tiles on an input side of a growth block. The three other TM simulations consist of tile types that are rotated versions of the north TM simulation shown. The halting tile types propagate one of the members of the tuple on which the TM halts, analogous to the border tile types. The bond types of TM tile types have a symbol from \mathcal{D} which indicates which simulation they belong to (omitted above).

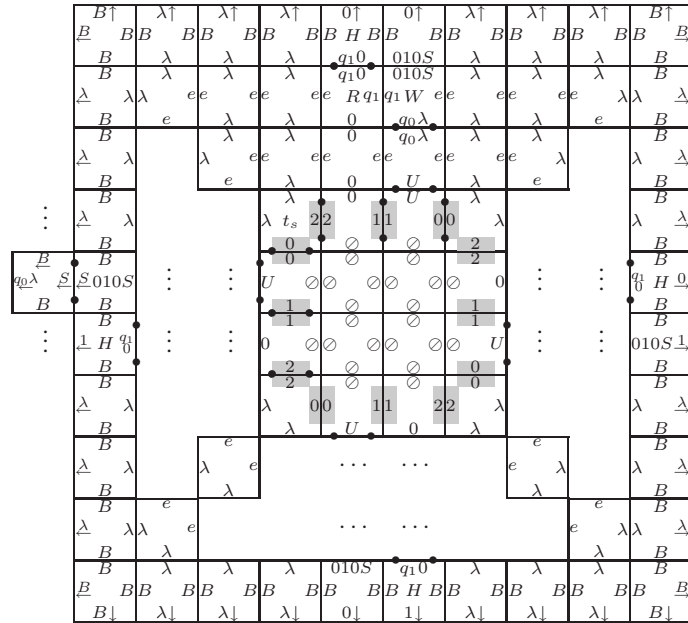
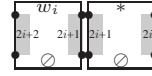


Figure 2.6: A simple seed block without unpacking showing the north TM simulation and the selective transmission of information through the borders. As shown, only the west side is a propagating output side; the other three sides are terminating output sides. All bond types in which a block type symbol is omitted have the block type symbol “ \star .” We assume that in bond types above, a single symbol $x \in \{0, 1, S, \lambda\}$ is the same as the tuple $(xxxx)$. The natural assembly sequence to consider is growing the seed frame first and then adding tiles row by row from the center (where a new row is started by the strength-2 bond).

a) Unpacking tile types for the north side of the seed frame:

We use n/k coding tiles in the input row, each encoding a binary substring (w_i) of length k . These tiles are interspersed with buffer tiles holding the symbol “*”. $\forall 0 \geq i \geq k/n - 1$:



The last tile of the seed row has symbol “U” which indicates the end of the input string.

To initiate the unpacking of new substrings: $\forall x \in \{0, 1\}^{k-1}, b \in \{0, 1\}$:



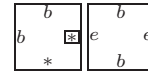
The following “extract bit” tile types perform the actual unpacking: $\forall j \in \{1, \dots, k-1\}, \forall x \in \{0, 1\}^j, b \in \{0, 1\}$:



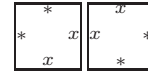
The following “copy remainder” tile types pass the remaining bits to the next extraction: $\forall j \in \{2, \dots, k-1\}, \forall x \in \{0, 1\}^j$:



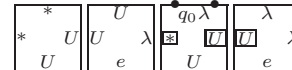
To copy a single bit in the last step of the unpacking of a substring and after unpacking every bit: $b \in \{0, 1\}$:



These tile types keep on copying substrings that are not yet being unpacked: $\forall x \in \{0, 1\}^k$:



Finally, the following tile types propagate the symbol “U”, which indicates the end of the input string, and initiate the TM simulation once the unpacking process finishes:



b) North unpacking example:

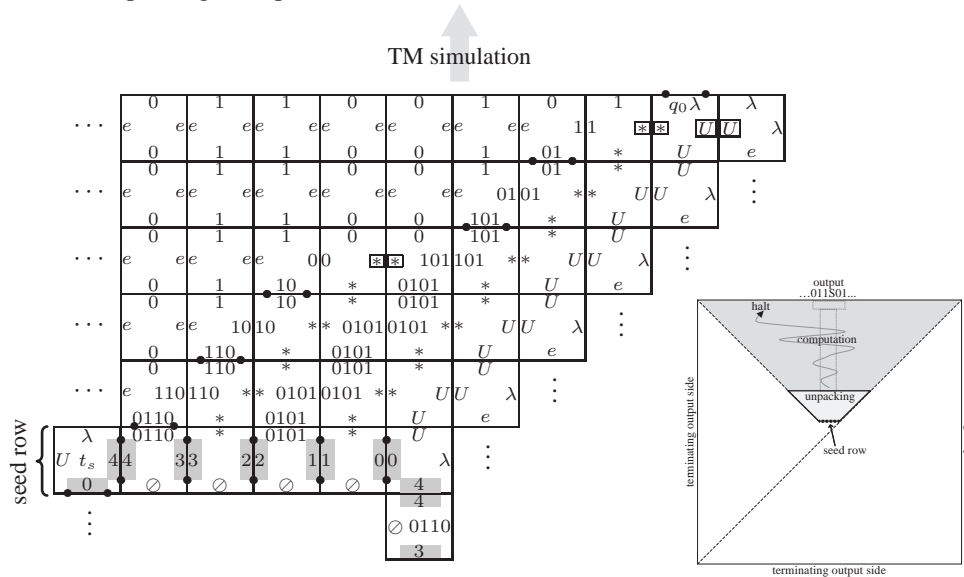


Figure 2.7: The unpacking for the north side of the seed frame. (a) The tile types used. (b) An example showing the unpacking of the string 01100101 if $k = 4$ for a seed block with up to four propagating output sides. Note that the unpacking process can be inserted immediately prior to the TM simulation without modifying other tile types. (inset) Internal structure of a seed block with only one propagating output side.

Altogether, a seed block with only one propagating output side requires only $15\frac{n}{\log n} + O(1)$ tile types. We will see in the next section that this is sufficient for growing any shape.

2.6.4 Programming Blocks and the Value of the Scaling Factor c

In order for our tile system to produce some assembly whose shape is \tilde{S} , instructions encoded in p must guide the construction of the blocks by deciding on which side of which block a new block begins to grow and what is encoded on the edge of each block. For our purposes, we take $p = p_{sb}\langle s \rangle$ (i.e., p_{sb} takes s as input), where s is a program that outputs the list of locations in the shape S . p_{sb} runs s to obtain this list and plans out a spanning tree t over these locations (it can just do a depth-first search) starting from some arbitrarily chosen location that will correspond to the seed block.* The information passed along the arrows in Fig. 2.1(b) is $p_{gb}\langle t, (i, j) \rangle$ which is the concatenation of a program p_{gb} to be executed within each growth block, and an encoding of the tree t and the location (i, j) of the block into which the arrow is heading. When executed, $p_{gb}\langle t, (i, j) \rangle$ evaluates to a 3-tuple encoding of $p_{gb}\langle t, D(i, j) \rangle$ together with symbol “ S ” for each propagating output side D . Thus, each growth block passes $p_{gb}\langle t, D(i, j) \rangle$ to its D^{th} propagating output side as directed by t . Note that program p_{sb} in the seed tile must also run long enough to ensure that c is large enough that the computation in the growth blocks has enough space to finish without running into the sides of the block or into the diagonal. Nevertheless, the scaling factor c is dominated by the building of t in the seed block, as the computation in the growth blocks takes only $poly(|S|)$.[†] Since the building of t is dominated by the running time of s , we have $c = poly(time(s))$.

2.6.5 Uniqueness of the Terminal Assembly



By Theorem 2.3.1 it is enough to demonstrate a locally deterministic assembly sequence ending in our target terminal assembly to be assured that this terminal assembly is uniquely produced. Consider the assembly sequence \tilde{A} in which the assembly is constructed *block by block* such that every block is finished before the next one started and each block is constructed by the natural assembly sequence described in the captions to Figs. 2.4 and 2.6. It is enough to confirm that in this natural assembly sequence every tile addition satisfies the definition of local determinism (Definition 2.3.2). It is easy to confirm that every tile not adjacent to a terminal output side of a block indeed satisfies these conditions. Other than on a terminal output side of a block (and on *null* tiles) there are no *termsides*: every side is either an *inputside* or a *proposide*. In our construction, each new tile binds through either a single strength-2 bond or two strength-1 bonds (thus condition 1 is satisfied since $\tau = 2$) such that no other tile type can bind through these *inputsides* (condition 2 is satisfied if the tile has no *termsides*). Note that inadvertent binding of a tile type from a different block type is prevented by the block type symbols.

Now let’s consider *termsides* around the terminal output sides of blocks (Fig. 2.8(a)). Here block type symbols come to the rescue again and prevent inadvertent binding. Let $t \in A$ be a tile with a *termside* (t can be *null*). We claim that $\forall t'$ s.t. $type(t') \in T$ and $pos(t') = pos(t)$, if $\Gamma_{termsides\tilde{A}(t)}^A(t') > 0$ then $\Gamma_{D-proposides\tilde{A}(t)}^A(t') < \tau = 2$. In other words, if t' binds on a *termside* of t , then it cannot bind strongly enough to violate local determinism, implying we can ignore *termsides*. Figure 2.8(a) shows in dotted lines the *termsides* that could potentially be involved in bonding. These *termsides* cannot have a strength-2 bond because symbol “ S ” is not propagated to terminal output sides of blocks. Thus t' binding only on a single *termside* of t is not enough. Can t' bind on two *termsides* of t ? To do so, it must be in a corner between two blocks, binding two terminal output sides of different blocks. But to bind in this way would require t' to bond to the block type symbol pattern[‡] shown in Fig. 2.8(b) (or its rotation), which none of the tile types in

*We can opt to always choose a leaf, in which case the seed block requires only one propagating output side. In this case the multiplicative factor a_1 is $15 + \varepsilon$, although the tile set used will depend upon the direction of growth from the leaf.

[†]Note that fewer than n rows are necessary to unpack a string of length n (Section 2.6.3). Since we can presume that p_{sb} reads its entire input and the universal TM needs to read the entire input program to execute it, the number of rows required for the unpacking process can be ignored with respect to the asymptotics of the scaling factor c .

[‡]The block type symbol pattern of a tile type consists of the block type symbols among its four bond types. For instance, the tile

type  has block type symbol pattern . If two bond types do not have matching block type symbols then obviously they cannot bind.

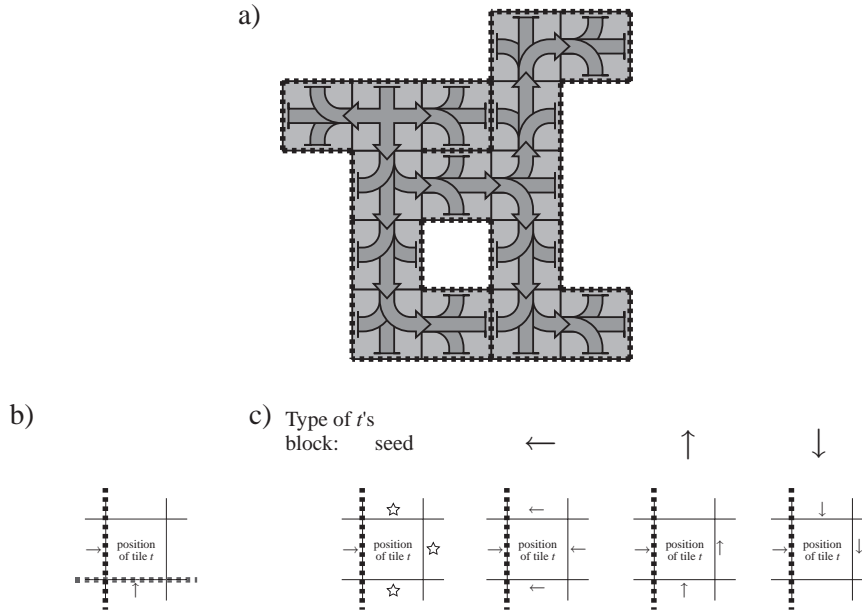


Figure 2.8: (a) The target terminal assembly with the dotted lines indicating the edges that have *termsides* with non-null bonds. (b) The block type symbols of adjacent tiles on two *termsides* of t (west and south in this case). (c) The block type symbols of adjacent tiles on a *termside* (west side in this case) and an *inputside* of t . If t is in the seed block or \leftarrow growth block, then the north, east, south sides may be the *inputsides*. If t is in a \uparrow block then the east and south sides may be the *inputsides*. If t is in a \downarrow block then the north and east sides may be the *inputsides*.

our tile system can do. Can t' bind on one *termside* and one *inputside* of t ? Say the *termside* of t that t' binds on is the west side (Fig. 2.8(c)). The tile to the west of t must be on the east terminal output side of a block, and thus it has symbol “ \rightarrow ” on its east side. So t' must have “ \rightarrow ” on the west, and depending on the type of block t is in, one of the other block type symbols as shown in Fig. 2.8(c). But again none of the tile types in our tile system has the necessary block type symbol pattern.

2.7 Generalizations of Shape Complexity

In this work we have established both upper and lower bounds relating the descriptive complexity of a shape to the number of tile types needed to self-assemble the shape within the standard Tile Assembly Model. The relationship is dependent upon a particular definition of shape that ignores its size. Disregarding scale in self-assembly appears to play a similar role as disregarding time in theories of computability and decidability. Those theories earned their universal standing by being shown to be identical for all “reasonable” models of computation. To what extent do our results depend on the particular model of self-assembly? Can one define a complexity theory for families of shapes in which the absolute scale is the critical resource being measured? In this section we discuss the generality and limitations of our result.

2.7.1 Optimizing the Main Result (Section 2.5)

Since the Kolmogorov complexity of a string depends on the universal Turing machine chosen, the complexity community adopted a notion of additive equivalence, where additive constants are ignored. However, Theorem 2.5.1 includes multiplicative constants as well, which are not customarily discounted. It might be possible to use a more clever method of unpacking (Section 2.6.2) and a seed block construction that reduces

the multiplicative constant a_1 of Theorem 2.5.1. Correspondingly, there might be a more efficient way to encode any tile system than that described in the proof of the theorem, and thereby increase a_0 .

Recall that s is the program for U producing the target coordinated shape S as a list of locations. For cases where our results are of interest, the scaling factor $c = \text{poly}(\text{time}(s))$ is extremely large, since $|S|$ is presumably enormous and s must output every location in S . The program s' that given (i, j) outputs 0/1 indicating whether S contains that location may run much faster than s for large shapes. Can our construction be adapted to use s' in each block rather than s in the seed block to obtain smaller scale? The problem with doing this directly is that the scale of the blocks, which sets the maximum allowed running time of computation in each block, must be set in the seed block. As a result, there must be some computable time bound on s' that is given to the seed block.

For any particular shape, there must be a range of achievable parameters: the number of tile types and the scaling factor. We know that we can obtain scaling factor 1 by using a unique tile type for each location. On the other extreme is our block construction which allows us to obtain an asymptotically optimal number of tile types at the expense of an enormous scaling factor. Presumably there is a gradual tradeoff between the number of tile types and the scale that can be achieved by a range of tile systems. The characterization of this tradeoff is a topic for future study.

In this vein, an important open problem remains of determining lower bounds on the scales of shapes produced by tile systems with an asymptotically optimal number of tile types. As an initial result of this kind, consider the following proof that an arbitrarily large scaling factor may need to be used if we stick to asymptotically optimal tile systems. Consider the coordinated shape that is a rectangle of width m and height 1. Clearly, it is an instance of the following shape \tilde{S} : a long, thin rectangle that is m times as long as it is high. According to Aggarwal et al. [4], the number of tile types required to self-assemble a long, thin rectangle that is n tiles long and k tiles high is $\Omega(\frac{n^{1/k}}{k})$. This implies that to produce any coordinated instance of \tilde{S} at scale c requires $|T| = \Omega(\frac{(mc)^{1/c}}{c})$ tile types. Now we can define what an asymptotically optimal tile system means for us by choosing a_1, b_1 and requiring that the number of tile types $|T|$ satisfy $|T| \log |T| \leq a_1 K(\tilde{S}) + b_1$. Since $K(\tilde{S}) = O(\log m)$, it follows through simple algebra that no matter what a_1, b_1 are, for large enough m , the scaling factor c needs to get arbitrarily large to avoid a contradiction.

2.7.2 Strength Functions

In most previous works on self-assembly, as in this work, strength functions are restricted with the following properties: (1) the effect that one tile has on another is equal to the effect that the other has on the first, i.e., g is *symmetric*: $g(\sigma, \sigma') = g(\sigma', \sigma)$; (2) the lack of an interaction is normalized to zero, i.e., $g(\sigma, \text{null}) = 0$; (3) there are no ‘‘adverse’’ interactions counteracting other interactions, i.e., g is *non-negative*; (4) only sides with matching bond types interact, i.e., g is *diagonal*: $g(\sigma, \sigma') = 0$ if $\sigma \neq \sigma'$.

Properties 1 and 2 seem natural enough. Our results are independent of property 3 because the encoding used for the lower bound of Theorem 2.5.1 is valid for strength functions taking on negative values. Property 4, which reflects the roots of the Tile Assembly Model in the Wang tiling model, is essential for the quantitative relationship expressed in Theorem 2.5.1: recent work by Aggarwal et al. [4] shows that permitting non-diagonal strength functions allows information to be encoded more compactly. Indeed, if property 4 is relaxed then replacing our unpacking process with the method of encoding used in that work and using Aggarwal et al.’s lower bound leads to the following form of Theorem 2.5.1: Assuming the maximum threshold τ is bounded by a constant, there exist constants a_0, b_0, a_1, b_1 such that for any shape \tilde{S} ,

$$a_0 K(\tilde{S}) + b_0 \leq \left(K_{sa}^{nd}(\tilde{S}) \right)^2 \leq a_1 K(\tilde{S}) + b_1$$

where K_{sa}^{nd} is the tile-complexity when non-diagonal strength functions are allowed. It is an open question whether the constant bound on τ can be relaxed.

2.7.3 Wang Tiling vs Self-Assembly of Shapes

Suppose one is solely concerned with the existence of a configuration in which all sides match, and not with the process of assembly. This is the view of classical tiling theory [7]. Since finite tile sets can enforce

uncomputable tilings of the plane [8, 16], one might expect greater computational power when the existence, rather than production, of a tiling is used to specify shapes. In this section we develop the notion of shapes in the Wang tile model [20] and show that results almost identical to the Tile Assembly Model hold. One conclusion of this analysis is that making a shape “practically constructible” (i.e., in the sense of the Tile Assembly Model) does not necessitate an increase in tile-complexity.

We translate the classic notion of the origin-restricted Wang tiling problem* as follows. An **(origin-restricted) Wang tiling system** is a pair (T, t_s) where T is a set of tile types and t_s is a **seed tile** with $\text{type}(t_s) \in T$. A configuration A is a valid tiling if all sides match and it contains the seed tile. Formally, A is a **valid tiling** if $\forall (i, j) \in \mathbb{Z}^2, D \in \mathcal{D}$, (1) $\text{type}(A(i, j)) \in T$, (2) $t_s \in A$, (3) $\text{bond}_D(A(i, j)) = \text{bond}_{D^{-1}}(A(D(i, j)))$.

Since valid tilings are infinite objects, how can they define finite coordinated shapes? For tile sets containing the *empty* tile type, we can define shapes analogously to the Tile Assembly Model. However, we cannot simply define the coordinated shape of a valid tiling to be the set of locations of non-empty tiles. For one thing, the set of non-empty tiles can be disconnected, unlike in self-assembly where any produced assembly is a single connected component. So we take the coordinated shape S_A of a valid tiling A to be the smallest region of non-*empty* tiles containing t_s that can be extended to infinity by *empty* tiles. Formally, S_A is the coordinated shape of the smallest subset of A that is a valid tiling containing t_s . If S_A is finite, then it is the **coordinated shape of valid tiling** A .[†] Shape \tilde{S} is the **shape of a valid tiling** A if $S_A \in \tilde{S}$.

Produced assemblies of a tile system (T, t_s, g, τ) are not necessarily valid tilings of Wang tiling system (T, t_s) because the Tile Assembly Model allows mismatching sides. Further, valid tilings of (T, t_s) are not necessarily produced assemblies of (T, t_s, g, τ) . Even if one considers only valid tilings that are connected components, there might not be any sequence of legal tile additions that assembles these configurations. Nonetheless, if a tile system uniquely produces a valid tiling A , then all valid tilings of the corresponding Wang tile system agree with A and have the same coordinated shape as A :

Lemma 2.7.1. *If $\text{empty} \in T$ and the tile system $\mathbf{T} = (T, t_s, g, \tau)$ uniquely produces assembly A such that A is a valid tiling of the Wang tiling system (T, t_s) then for all valid tilings A' : (1) $\forall (i, j) \in \mathbb{Z}^2, \text{type}(A(i, j)) \neq \text{empty} \Rightarrow A'(i, j) = A(i, j)$, (2) $S_{A'} = S_A$.*

Proof. Consider an assembly sequence \vec{A} of \mathbf{T} ending in assembly A and let A' be a valid tiling of (T, t_s) . Suppose t_n is the first tile added in this sequence such that $t' = A'(\text{pos}(t_n)) \neq t_n$. Since A' is a valid tiling, t' must match on all sides, including *inputsides* $\vec{A}(t_n)$. But this implies that two different tiles can be added in the same location in \vec{A} which means that A is not uniquely produced. This implies part (1) of the lemma. Now, to be a valid tiling, all exposed sides of assembly A must be null. Thus if A' and A agree on all places where A is non-*empty*, then $S_{A'} = S_A$ and part (2) of the lemma follows. \square

Define the **tile-complexity** K_{wt} of a shape \tilde{S} in the origin-restricted Wang tiling model as the minimal number of tile types in a Wang tiling system with the property that a valid tiling exists and there is a coordinated shape $S \in \tilde{S}$ such that for every valid tiling A , $S_A = S$.

Theorem 2.7.1. *There exist constants a_0, b_0, a_1, b_1 such that for any shape \tilde{S} ,*

$$a_0 K(\tilde{S}) + b_0 \leq K_{wt}(\tilde{S}) \log K_{wt}(\tilde{S}) \leq a_1 K(\tilde{S}) + b_1.$$

Proof. (Sketch) The left inequality follows in a manner similar to the proof of Theorem 2.5.1. Suppose every valid tiling of our Wang tile system has coordinated shape S . Any Wang tiling system of n tile types can be represented using $O(n \log n)$ bits. Making use of this information as input, we can use a constant-size program to find, through exhaustive search, the smallest region containing t_s surrounded by *null* bond types in some valid tiling. Thus, $O(n \log n)$ bits are enough to compute an instance of \tilde{S} . To prove the right inequality, our original block construction almost works, except that there are mismatches between a terminal output side of a block and the abutting terminal output side of the adjacent block or the surrounding *empty* tiles (i.e., along the dotted lines in Fig. 2.8(a)). Consequently, the original construction does not yield a valid tiling. Nonetheless, a minor variant of our construction overcomes this problem. Instead of relying on

*The *unrestricted* Wang tile model does not have a seed tile [20, 5, 18].

[†] S_A can be finite only if *empty* $\in T$ because otherwise no configuration containing an *empty* tile can be a valid tiling.

mismatching bond type symbols to prevent inadvertent binding to terminal output sides of blocks, we can add an explicit capping layer that covers the terminal output sides with *null* bond types but propagates information through propagating output sides. This way, the terminal output sides of blocks are covered by *null* bond types and match the terminal output sides of the adjacent block and *empty* tiles. These modifications can be made preserving local determinism, which, by Lemma 2.7.1, establishes that the coordinated shape of any valid tiling is an instance of \tilde{S} . \square

There may still be differences in the computational power between Wang tilings and self-assembly processes. For example, consider the smallest Wang tiling system and the smallest self-assembly tile system that produce instances of \tilde{S} . The instance produced by the Wang tiling system might be much smaller than the instance produced by self-assembly. Likewise, there might be *coordinated* shapes that can be produced with significantly fewer tile types by a Wang tiling system than by a self-assembly system.

Keep in mind that the definition we use for saying when a Wang tiling system produces a shape was chosen as a natural parallel to the definition used for self-assembly, but alternative definitions may highlight other interesting phenomena specific to Wang tilings. For example, one might partition tiles into two subsets, “solution” and “substance” tiles, and declare shapes to be connected components of substance tiles within valid tilings. In such tilings — reminiscent of “vicinal water” in chemistry — the solution potentially can have a significant (even computational) influence that restricts possible shapes of the substance, and hence the size of produced shapes needn’t be so large as to contain the full computation required to specify the shape.

2.7.4 Sets of Shapes

Any coordinated shape S can be trivially produced by a self-assembly tile system or by a Wang tiling of $|S|$ tile types. Interesting behavior occurs only when the number of tile types is somehow restricted and the system is forced to perform some non-trivial computation to produce a shape. Previously in this paper, we restricted the number of tile types in the sense that we ask what is the minimal number of tile types that can produce a given shape. We saw that ignoring scale in this setting allows for an elegant theory. In the following two sections the restriction on the number of tile types is provided by the infinity of shapes they must be able to produce. Here we will see as well that ignoring scale allows for an elegant theory.

Adleman [2] asks “What are the ‘assemblable [*sic*] shapes?’ — (analogous to what are the ‘computable functions’)?” While this is still an open question for coordinated shapes, our definition of a shape ignoring scale and translation leads to an elegant answer. A set of binary strings \tilde{L} is a language of shapes if it consists of (standard binary) encodings of lists of locations that are coordinated shapes in some set of shapes: $\tilde{L} = \{ \langle S \rangle \text{ s.t. } S \in \tilde{S} \text{ and } \tilde{S} \in R \}$ for some set of shapes R . Note that every instance of every shape in R is in this language. The language of shapes \tilde{L} is recursively enumerable if there exists a Turing machine that halts upon receiving $\langle S \rangle \in \tilde{L}$, and does not halt otherwise. We say a tile system \mathbf{T} produces the language of shapes \tilde{L} if $\tilde{L} = \{ \langle S \rangle \text{ s.t. } S \in \tilde{S}_A \text{ for some } A \in Term(\mathbf{T}) \}$. We may want \tilde{L} to be *uniquely produced* in the sense that the $A \in Term(\mathbf{T})$ is unique for each shape. Further, to prevent infinite spurious growth we may also require \mathbf{T} to satisfy the *non-cancerous* property: $\forall B \in Prod(\mathbf{T}), \exists A \in Term(\mathbf{T}) \text{ s.t. } B \rightarrow_{\mathbf{T}}^* A$. The following lemma is valid whether or not these restrictions are made.

Lemma 2.7.2. *A language of shapes \tilde{L} is recursively enumerable if and only if it is (uniquely) produced by a (non-cancerous) tile system.*

Proof. (Sketch) First of all, for any tile system \mathbf{T} we can make a TM that given a coordinated shape S as a list of locations, starts simulating all possible assembly sequences of \mathbf{T} and halts iff it finds a terminal assembly that has shape \tilde{S} . Therefore, if \tilde{L} is produced by a tile system, \tilde{L} is recursively enumerable. In the other direction, if \tilde{L} is recursively enumerable then there is a program p that given n outputs the n^{th} shape from \tilde{L} (in some order) without repetitions. Our programmable block construction can be modified to execute a non-deterministic universal TM in the seed block by having multiple possible state transitions. We make a program that non-deterministically guesses n , feeds it to p , and proceeds to build the returned shape. Note that since every computation path terminates, this tile system is non-cancerous, and since p enumerates without repetitions, the language of shapes is uniquely produced. \square

Note that the above lemma does not hold for languages of coordinated shapes, defined analogously. Many simple recursively enumerable languages of coordinated shapes cannot be produced by any tile system. For example, consider the language of equilateral width-1 crosses centered at $(0, 0)$. No tile system produces this language. Scale equivalence is crucial because it allows arbitrary amounts of information to be passed between different parts of a shape; otherwise, the amount of information is limited by the width of a shape.

The same lemma can be attained for the Wang tiling model in an analogous manner using the construction from Section 2.7.3. Let us say a Wang tiling system (T, t_s) produces the language of shapes \tilde{L} if $\tilde{L} = \{\langle S \rangle \mid S \in \tilde{S}_A \text{ for some valid tiling } A \text{ of } (T, t_s)\}$. Analogously to tile systems, we may require the *unique production* property that there is exactly one such A for each shape. Likewise, corresponding to the non-cancerous property of tile systems, we may also require the tiling system to have the *non-cancerous* property that every valid tiling has a coordinated shape (i.e., is finite). Again, the following lemma is true whether or not these restrictions are made.

Lemma 2.7.3. *A language of shapes \tilde{L} is recursively enumerable if and only if it is (uniquely) produced by a (non-cancerous) Wang tiling system.*

2.7.5 Scale Complexity of Shape Functions

Expanding upon the notion of a shape being the output of a universal computation process as mentioned in the Introduction, let us consider tile systems effectively computing a function from binary strings to shapes. The universal “programmable block” constructor presented in Section 2.6 may be taken as an example of such a tile set, if the full seed block is considered as an initial seed assembly rather than as part of the tile set per se. In this case, the remaining tile set is of constant size, and will construct an arbitrary algorithmic shape when presented with a seed assembly containing the relevant program. The universal constructor tile set’s efficiency, then, can be measured in terms of the scale of the produced shape. Similarly, other “programmable” tile sets may produce a limited set of shapes, but potentially with greater efficiency. (Such tile sets can be thought to produce a language of shapes (Section 2.7.4) such that the choice of the produced shape can be deterministically specified.) For tile systems outputting shapes in this manner, we can show that the total number of tiles (not tile *types*) in the produced shape is closely connected to the time complexity of the corresponding function from binary strings to shapes in terms of Turing machines. The equivalent connection can be made between non-deterministic Turing machines and the size of valid tilings in the Wang tiling model.

Let f be a function from binary strings to shapes. We say that a Turing machine M computes this function if for all x , $f(x) = \tilde{S} \Leftrightarrow \exists S \in \tilde{S} \text{ s.t. } M(x) = \langle S \rangle$. The standard notion of time-complexity applies: $f \in TIME_{TM}(t(n))$ if there is a TM computing it running in time bounded by $t(n)$ where n is the size of the input. In Section 2.6.2.2 we saw how binary input can be provided to a tile system via a seed frame wherein all four sides of a square present the bitstring. Let us apply this convention here.* Extending the notion of the seed in self-assembly to the entire seed frame and using this as the input for a computation [17], we say a tile system computes f if: [starting with the seed frame encoding x the tile system uniquely produces an assembly of shape \tilde{S}] iff $f(x) = \tilde{S}$. We say that $f \in TILES_{SA}(t(n))$ if there is a tile system computing it and the size of coordinated shapes produced (in terms of the number of non-empty locations) for inputs of size n is upper bounded by $t(n)$. Similar definitions can be made for non-deterministic Turing machines and Wang tiling systems. We say that a NDTM N computes f if: [every computation path of N on input x ending in an accept state (as opposed to a reject state) outputs $\langle S \rangle$ for some $S \in \tilde{S}$] iff $f(x) = \tilde{S}$. For non-deterministic Turing machines, $f \in TIME_{NDTM}(t(n))$ if there is a NDTM computing f such that every computation path halts after $t(n)$ steps. Extending the notion of the seed for Wang tilings to the entire seed frame as well, we say a Wang tiling system computes f if: all valid tilings containing the seed frame have a coordinated shape and this coordinated shape is the same for all such valid tilings, and it is an instance of the shape $f(x)$. We say that $f \in TILES_{WT}(t(n))$ if there is a tiling system computing it and the size of coordinated shapes produced for inputs of size n is upper bounded by $t(n)$.

Theorem 2.7.2. (a) *If $f \in TILES_{SA}(t(n))$ then $f \in TIME_{TM}(O(t(n)^4))$*

* Any other similar method would do. For the purposes of this section, it does not matter whether we use the one bit per tile encoding or the encoding requiring unpacking (Section 2.6.3).

- (b) If $f \in TIME_{TM}(t(n))$ then $f \in TILES_{SA}(O(t(n)^3))$
- (c) If $f \in TILES_{WT}(t(n))$ then $f \in TIME_{NDTM}(O(t(n)^4))$
- (d) If $f \in TIME_{NDTM}(t(n))$ then $f \in TILES_{WT}(O(t(n)^3))$

Proof. (Sketch) (a) Let \mathbf{T} be a tile system computing f such that the total number of tiles used on an input of size n is $t(n)$. A Turing machine with a 2-D tape can simulate the self-assembly process of \mathbf{T} with an input of size n in $O(t(n)^2)$ time: for each of the $t(n)$ tile additions, it needs to search $O(t(n))$ locations for the next addition. This 2-D Turing machine can be simulated by a regular Turing machine with a quadratic slowdown.*

(b) Let M be a deterministic Turing machine that computes f and runs in time $t(n)$. Instead of simulating a universal Turing machine in the block construction, we simulate a Turing machine M' which runs M on input x encoded in the seed frame and acts as program p_{sb} in Section 2.6.4. Then the scale of each block is $O(t(n))$, which implies that each block consists of $O(t(n)^2)$ tiles. Now the total number of blocks cannot be more than the running time of M since M outputs every location that corresponds to a block. Thus the total number of tiles is $O(t(n)^3)$.

(c) A similar argument applies to the Wang tiling system as (a) with the following exception. A Wang tiling system can simulate a non-deterministic Turing machine and still be able to output a unique shape. The tiling system can be designed such that if a reject state is reached, the tiling cannot be a valid tiling. For example, the tile representing the reject state can have a bond type that no other tile matches. Thus all valid tilings correspond to accepting computations.

(d) Simulation of Wang tiling systems can, in turn, be done by a non-deterministic Turing machine as follows. Suppose every valid tiling of our Wang tile system has coordinated shape S . The simulating NDTM acts similar to the TM simulating self-assembly above, except that every time two or more different tiles can be added in the same location, it non-deterministically chooses one. If the NDTM finds a region containing the seed frame surrounded by *null* bond types, it outputs the shape of the smallest such region and enters an accept state. Otherwise, at some point no compatible tile can be added, and the NDTM enters a reject state. The running time of accepting computations is $O(t(n)^2)$ via the same argument as for (b). \square

If, as is widely believed, NDTMs can compute some functions in polynomial time that require exponential time on a TM, then it follows that there exist functions from binary strings to shapes that can be computed much more efficiently by Wang tiling systems than by self-assembly, where efficiency is defined in terms of the size of the coordinated shape produced.

The above relationship between *TIME* and *TILES* may not be the tightest possible. As an alternative approach, very small-scale shapes can be created as Wang tilings by using an NDTM that recognizes tuples (i, j, x) , rather than one that generates the full shape. This will often yield a compact construction. As a simple example, this approach can be applied to generating circles with radius x at scale $O(n^2)$ where $n = O(\log x)$. It remains an open question how efficiently circles can be generated by self-assembly.

2.7.6 Other Uses of Programmable Growth

The programmable block construction is a general way of guiding the large scale growth of the self-assembly process and may have applications beyond those explored so far. For instance, instead of constructing shapes, the block construction can be used to simulate other tile systems in a scaled manner using fewer tile types. It is easy to reprogram it to simulate, using few tile types, a large deterministic $\tau = 1$ tile system for which a short algorithmic description of the tile set exists. We expect a slightly extended version of the block construction can also be used to provide compact tile sets that simulate other $\tau = 2$ tile systems that have short algorithmic descriptions.

To self-assemble a circuit, it may be that the shape of the produced complex is not the correct notion. Rather one may consider finite patterns, where each location in a shape is “colored” (e.g. resistor, transistor,

*The rectangular region of the 2-D tape previously visited by the 2-D head (the arena) is represented row by row on a 1-D tape separated by special markers. The current position of the 2-D head is also represented by a special marker. If the arena is $l \times m$, a single move of the 2-D machines which does not escape the current arena requires at most $O(m^2)$ steps, while a move that escapes it in the worst case requires an extra $O(ml^2)$ steps to increase the arena size. We have $m, l = O(t(n))$, and the number of times the arena has to be expanded is at most $O(t(n))$.

wire, etc.). Further, assemblies that can grow arbitrarily large may be related to infinite patterns. What is the natural way to define the self-assembly complexity of such patterns? Do our results (Section 2.5) still hold?

2.8 Appendix

2.8.1 Local Determinism Guarantees Unique Production: Proof of Theorem 2.3.1

Lemma 2.8.1. *If \vec{A} is a locally deterministic assembly sequence of \mathbf{T} , then for every assembly sequence \vec{A}' of \mathbf{T} and for every tile $t' = t'_n$ added in \vec{A}' the following conditions hold, where $t = A(\text{pos}(t'))$.*

- (i) $\text{inputsides}^{\vec{A}'}(t') = \text{inputsides}^{\vec{A}}(t)$,
- (ii) $t' = t$.

Proof. Suppose $t' = t'_n$ is the first tile added that fails to satisfy one of the above conditions. Consider any $D \in \text{inputsides}^{\vec{A}'}(t')$. Tile $t_D = A'(D(\text{pos}(t')))$ must have been added before t' in \vec{A}' and so $D^{-1} \notin \text{inputsides}^{\vec{A}'}(t_D) = \text{inputsides}^{\vec{A}}(t_D)$. This implies $D \notin \text{propsides}^{\vec{A}}(t)$ and so,

$$\text{inputsides}^{\vec{A}'}(t') \cap \text{propsides}^{\vec{A}}(t) = \emptyset. \quad (2.2)$$

Now, $\forall D, \Gamma_D^{A'_n}(t') \leq \Gamma_D^A(t)$ because A'_n has no more tiles than A and except at $\text{pos}(t)$ they all agree. eq. 2.2 implies

$$\Gamma_{\text{inputsides}^{\vec{A}'}(t')}^A(t') \leq \Gamma_{\mathcal{D}-\text{propsides}^{\vec{A}}(t)}^A(t').$$

Therefore,

$$\Gamma_{\text{inputsides}^{\vec{A}'}(t')}^{A'_n}(t') \leq \Gamma_{\mathcal{D}-\text{propsides}^{\vec{A}}(t)}^A(t').$$

So by property (2) of Definition 2.3.2, no tile of type $\neq \text{type}(t)$ could have been sufficiently bound here by $\text{inputsides}^{\vec{A}'}(t')$ and thus $t' = t$. Therefore, t' cannot fail the second condition (ii).

Now, suppose t' fails the first condition (i). Because of property (1) of Definition 2.3.2, this can only happen if $\exists D \in \text{inputsides}^{\vec{A}'}(t') - \text{inputsides}^{\vec{A}}(t')$. Since $D \notin \text{inputsides}^{\vec{A}}(t')$, t_D must have been added after t' in \vec{A} . So since t_D binds t' , $D^{-1} \in \text{inputsides}^{\vec{A}}(t_D)$ and so $D \in \text{propsides}^{\vec{A}}(t)$. But by eq. 2.2 this is impossible. Thus we conclude $A' \subseteq A$. \square

Lemma 2.8.1 directly implies that if there exists a locally deterministic assembly sequence \vec{A} of \mathbf{T} then $\forall A' \in \text{Prod}(\mathbf{T}), A' \subseteq A$. Theorem 2.3.1 immediately follows: If there exists a locally deterministic assembly sequence \vec{A} of \mathbf{T} then \mathbf{T} uniquely produces A .

Since local determinism is a property of the *inputsides* classification of tiles in a terminal assembly, Lemma 2.8.1 also implies:

Corollary 2.8.1. *If there exists a locally deterministic assembly sequence \vec{A} of \mathbf{T} then every assembly sequence ending in A is locally deterministic.*

2.8.2 Scale-Equivalence and “ \cong ” are Equivalence Relations

Translation-equivalence is clearly an equivalence relation. Let us write $S_0 \stackrel{tr}{=} S_1$ if the two coordinated shapes are translation equivalent.

Lemma 2.8.2. *If $S = S_0^d$ and $S_0 = S_m^k$ then $S = S_m^{dk}$.*

Proof. $S(i, j) = S_0(\lfloor i/d \rfloor, \lfloor j/d \rfloor) = S_m(\lfloor \lfloor i/d \rfloor / k \rfloor, \lfloor \lfloor j/d \rfloor / k \rfloor) = S_m(\lfloor i/dk \rfloor, \lfloor j/dk \rfloor)$. \square

Lemma 2.8.3. *If $S_0 \stackrel{tr}{=} S_1$ then $S_0^d \stackrel{tr}{=} S_1^d$.*

Proof. $S_0^d(i, j) = S_0(\lfloor i/d \rfloor, \lfloor j/d \rfloor) = S_1(\lfloor i/d \rfloor + \Delta i, \lfloor j/d \rfloor + \Delta j) = S_1(\lfloor \frac{i+d\Delta i}{d} \rfloor, \lfloor \frac{j+d\Delta j}{d} \rfloor) = S_1^d(i + d\Delta i, j + d\Delta j)$. \square

To show that scale equivalence is an equivalence relation, the only non-trivial property is transitivity. Suppose $S_0^c = S_1^d$ and $S_1^{d'} = S_2^{c'}$ for some $c, c', d, d' \in \mathbb{N}$. $(S_1^d)^{d'} = (S_1^{d'})^d = S_1^{d'd}$ by Lemma 2.8.2. Thus, $S_1^{d'd} = (S_0^c)^{d'} = (S_2^{c'})^d$, and by Lemma 2.8.2, $S_0^{cd'} = S_2^{c'd}$.

To show that “ \cong ” is an equivalence relation, again only transitivity is non-trivial. Suppose $S_0 \cong S_1$ and $S_1 \cong S_2$. In other words, $S_0^c \stackrel{tr}{=} S_1^d$ and $S_1^{d'} \stackrel{tr}{=} S_2^{c'}$ for some $c, c', d, d' \in \mathbb{N}$. By Lemma 2.8.3, $(S_0^c)^{d'} \stackrel{tr}{=} (S_1^d)^{d'}$ and $(S_1^{d'})^d \stackrel{tr}{=} (S_2^{c'})^d$. Then by Lemma 2.8.2, $S_0^{cd'} \stackrel{tr}{=} S_1^{d'd}$ and $S_1^{d'd} \stackrel{tr}{=} S_2^{c'd}$ which implies $S_0^{cd'} \stackrel{tr}{=} S_2^{c'd}$ by the transitivity of translation equivalence. In other words, $S_0 \cong S_2$.

Acknowledgments

We thank Len Adleman, members of his group, Ashish Goel, and Paul Rothmund for fruitful discussions and suggestions. We also thank Rebecca Schulman and David Zhang for useful and entertaining conversations about descriptonal complexity of tile systems, and an anonymous reviewer for a very careful reading of this paper and helpful comments.

Bibliography

- [1] L. Adleman, Q. Cheng, A. Goel, M.-D. Huang, D. Kempe, P. M. de Espanes, and P. W. K. Rothmund. Combinatorial optimization problems in self-assembly. In *ACM Symposium on Theory of Computing*, 2002.
- [2] L. M. Adleman. Toward a mathematical theory of self-assembly (extended abstract). Technical report, University of Southern California, 1999.
- [3] L. M. Adleman, Q. Cheng, A. Goel, and M.-D. A. Huang. Running time and program size for self-assembled squares. In *ACM Symposium on Theory of Computing*, pages 740–748, 2001.
- [4] G. Aggarwal, M. Goldwasser, M. Kao, and R. T. Schweller. Complexities for generalized models of self-assembly. In *Symposium on Discrete Algorithms*, 2004.
- [5] R. Berger. The undecidability of the domino problem. *Memoirs of the American Mathematical Society*, 66, 1966.
- [6] M. Cook, P. W. K. Rothmund, and E. Winfree. Self-assembled circuit patterns. In *DNA-Based Computers 9*, pages 91–107, 2004.
- [7] B. Grunbaum and G. Shephard. *Tilings and Patterns*. W.H. Freeman and Company, 1986.
- [8] W. Hanf. Nonrecursive tilings of the plane I. *The Journal of Symbolic Logic*, 39:283–285, 1974.
- [9] P. W. K. Rothmund. *Theory and Experiments in Algorithmic Self-Assembly*. PhD thesis, University of Southern California, 2001.
- [10] P. W. K. Rothmund, N. Papadakis, and E. Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology*, 2:e424, 2004.
- [11] P. W. K. Rothmund and E. Winfree. The program-size complexity of self-assembled squares (extended abstract). In *ACM Symposium on Theory of Computing*, pages 459–468, 2000.
- [12] T. H. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J. H. Reif, and N. C. Seeman. Construction, analysis, ligation, and self-assembly of DNA triple crossover complexes. *Journal of the American Chemical Society*, 122:1848–1860, 2000.

- [13] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, second edition, 1997.
- [14] C. Mao, T. H. LaBean, J. H. Reif, and N. C. Seeman. Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. *Nature*, 407:493–496, 2000.
- [15] C. Mao, W. Sun, and N. C. Seeman. Designed two-dimensional DNA Holliday junction arrays visualized by atomic force microscopy. *Journal of the American Chemical Society*, 121:5437–5443, 1999.
- [16] D. Myers. Nonrecursive tilings of the plane II. *The Journal of Symbolic Logic*, 39:286–294, 1974.
- [17] J. H. Reif. Local parallel biomolecular computation. In *DNA-Based Computers: III Proceedings of a DIMACS Workshop*, pages 217–254, 1997.
- [18] R. M. Robinson. Undecidability and nonperiodicity of tilings of the plane. *Inventiones Mathematicae*, 12:177–209, 1971.
- [19] J. von Neumann. *The Theory of Self Reproducing Automata*. University of Illinois Press, 1966.
- [20] H. Wang. Proving theorems by pattern recognition. II. *Bell Systems Technical Journal*, 40:1–42, 1961.
- [21] E. Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, 1998.
- [22] E. Winfree. Simulations of computing by self-assembly. Technical report, California Institute of Technology, 1998.
- [23] E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman. Design and self-assembly of two dimensional DNA crystals. *Nature*, 394:539–544, 1998.

Chapter 3

Complexity of Compact Proofreading for Self-Assembled Patterns

This chapter was published as: David Soloveichik and Erik Winfree, “Complexity of Compact Proofreading for Self-Assembled Patterns,” Proceedings of DNA Computing 11, Lecture Notes in Computer Science 3892: 305-324, 2006.

3.1 Abstract

Fault-tolerance is a critical issue for biochemical computation. Recent theoretical work on algorithmic self-assembly has shown that error-correcting tile sets are possible, and that they can achieve exponential decrease in error rates with a small increase in the number of tile types and the scale of the construction [24, 4]. Following [17], we consider the issue of applying similar schemes to achieve error correction without any increase in the scale of the assembled pattern. Using a new proofreading transformation, we show that compact proofreading can be performed for some patterns with a modest increase in the number of tile types. Other patterns appear to require an exponential number of tile types. A simple property of existing proofreading schemes — a strong kind of redundancy — is the culprit, suggesting that if general purpose compact proofreading schemes are to be found, this type of redundancy must be avoided.

3.2 Introduction

The Tile Assembly Model [22, 23] formalizes a generalized crystal growth process by which an organized structure can spontaneously form from simple parts. This model considers the growth of two dimensional “crystals” made out of square units called tiles. Typically, there are many types of tiles that must compete to bind to the crystal. A new tile can be added to a growing complex if it binds strongly enough. Each of the four sides of a tile has an associated bond type that interacts with matching sides of other tiles that have already been incorporated. The assembly starts from a specified seed assembly and proceeds by sequential addition of tiles. Tiles do not get used up since it is assumed there is an unbounded supply of tiles of each type. This model has been used to theoretically examine how to use self-assembly for massively parallel DNA computation [21, 26, 16, 13], for creating objects with programmable morphogenesis [10, 1, 2, 20], for patterning of components during nanofabrication of molecular electronic circuits [6], and for studying self-replication and Darwinian evolution of information-bearing crystals [18, 19]. Fig. 3.1 illustrates two different patterns and the corresponding tile systems that self-assemble into them. Both patterns are produced by similar tile systems using only two bond types, four tile types, simple Boolean rules and similar seed assemblies (the L-shaped boundaries).

Confirming the physical plausibility and relevance of the abstraction, several self-assembling systems have been demonstrated using DNA molecules as tiles, including both periodic [25, 15, 12] and algorithmic patterns [14, 9, 3]. A major stumbling block to making algorithmic self-assembly practical is the error rate

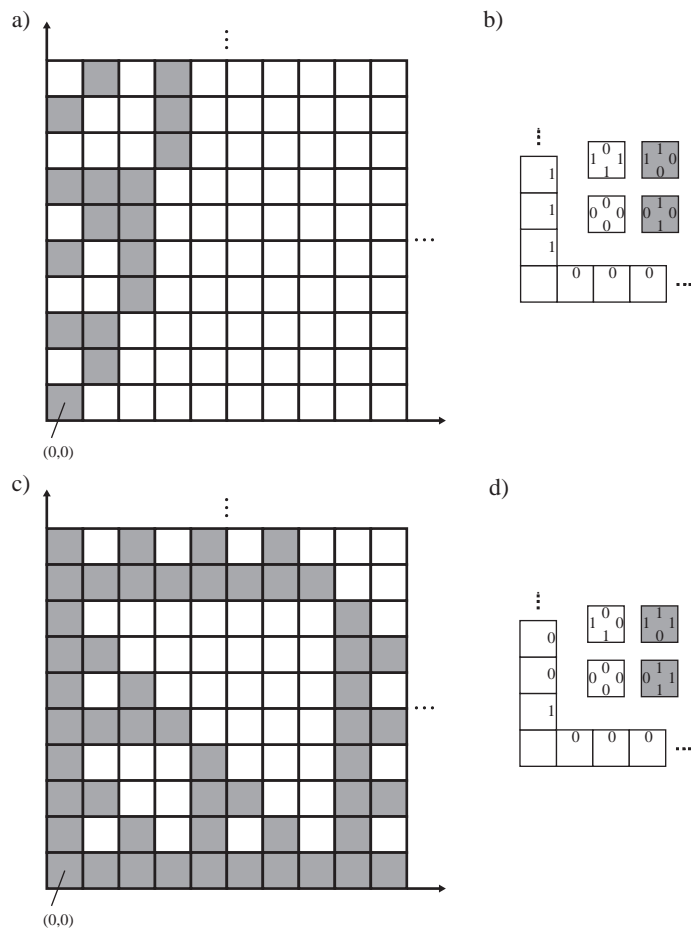


Figure 3.1: (a) A binary counter pattern and (b) a tile system constructing it. (c) A Sierpinski pattern and (d) a tile system constructing it. In this formalism, identically labeled sides match and tiles cannot be rotated. Tiles may attach to the growing assembly only if at least two sides match, i.e., if two bonds can form. Mismatches neither help nor hinder assembly. Note that the tile choice at each site is deterministic for these two tile sets.

inherent in any stochastic biochemical implementation. Current implementations seem to suffer error rates of 1% to 15% [9, 3]. This means that on average every eighth to hundredth tile that is incorporated does not correctly bond with its neighbors. Once such a mistake occurs, the erroneous information can be propagated to tiles that are subsequently attached. Thus, a single mistake can result in a drastically different pattern being produced. With this error rate, structures of size larger than roughly 100 tiles cannot be assembled reliably.

There are generally two ways to improve the error-robustness of the assembly process. First, the physics of the process can be modified to achieve a lower probability of the incorporation of incorrect tiles into the growing complex. The second method, which we pursue here, is to use some logical properties of the tiles to perform error correction.

Proofreading tile sets for algorithmic self-assembly were introduced by Winfree and Bekobolotov [24]. The essential idea was to make use of a redundant encoding of information distributed across k tiles, making isolated errors impossible: to continue growth, errors must appear in multiples of k . Thanks to the reversible nature of crystallization, growth from erroneous tiles stalls and the erroneous tiles subsequently dissociate, allowing another chance for correct growth. Using this approach, a large class of tile sets can be transformed into more robust tile sets that assemble according to the same logic.

However (a) the proofreading tile sets produce assemblies k times larger than the original tile sets, involving k^2 times as many tiles; and (b) the improvement in error rates did not scale well with k in simulation. Chen and Goel [4] developed *snaked proofreading tile sets* that generalize the proofreading construction in a

way that further inhibits growth on crystal facets. They were able to prove, with respect to a reversible model of algorithmic self-assembly, that error rates decrease exponentially with k , and thus to make an $N \times N$ pattern required only $k = \Omega(\log N)$. This provides a solution for (b), although the question of optimality remains open. Reif et al. [17] raised the question of whether more compact proofreading schemes could be developed, and showed how to transform the two tiles sets shown in Fig. 3.1 to obtain lower error rates without any sacrifice in scale. However, Reif et al. did not give a general construction that works for any original tile set, and did not analyze how the number of tile types would scale if the construction were to be generalized to obtain greater degrees of proofreading. Thus, question (a) concerning whether this can be improved in general and at what cost remained open.

The question of compactness is particularly important when self-assembly is used for molecular fabrication tasks, in which case the scale of the final pattern is of direct and critical importance. Furthermore, the question of scale is a fundamental issue for the theory of algorithmic self-assembly. In the error-free case, disregarding scale can drastically change the minimal number of tile types required to produce a given shape (Chapter 2); some shapes can be assembled from few tile types at a small scale, while other shapes can *only* be assembled from few tile types at a large scale. Examining whether proofreading can be performed without sacrificing scale is both of practical significance and could lead to important theoretical distinctions.

If it is the case that some patterns can't be assembled with low error rates at the original scale using a concise tile set, while for other patterns compact proofreading can be done effectively, then we would be justified in calling the former intrinsically fragile, and the latter intrinsically robust. Any such distinctions should be independent of any particular proofreading scheme. Indeed, we here show that this is true (in a certain sense), and we give a combinatorial criterion that distinguishes fragile patterns from robust patterns. As examples, we show that the two patterns discussed in Reif et al.'s work on compact proofreading [17] and shown in Fig. 3.1 are fundamentally different, in that (within a wide class of potential proofreading schemes considered here) the cost of obtaining reliable assembly at the same scale becomes dramatically different as lower error rates are required.

3.2.1 The Abstract Tile Assembly Model

This section informally summarizes the abstract Tile Assembly Model (aTAM). See [8], Chapter 2 for a formal treatment. Self-assembly occurs on a $\mathbb{Z} \times \mathbb{Z}$ grid of unit square locations, on which unit-square **tiles** may be placed under specific conditions. Each tile has **bond types** on its north, east, south, and west sides. A finite set of **tile types** defines the set of possible tiles that can be placed on the grid. Tile types are oriented and therefore a rotated version of a tile type is considered to be a different tile type. A single tile type may be used an arbitrary number of times. A **configuration** is a set of tiles such that there is at most one tile in every location $(i, j) \in \mathbb{Z} \times \mathbb{Z}$. Two adjacent tiles **bond** if their abutting sides have matching bond types. Further, each bond type forms bonds of a specific strength, called its **interaction strength**. In this paper the three possible strengths of bonds are $\{0, 1, 2\}$. A new tile can be added to an empty spot in a configuration if and only if the sum of its interaction strengths with its neighbors reaches or exceeds some parameter τ . The tile systems shown in this paper use $\tau = 2$, i.e., at least a single strong (strength 2) or two weak (strength 1) bonds are needed to secure a tile in place.

For the purposes of this paper, a tile system consists of a finite set of tile types T with specific interaction strengths associated with each bond type, and a start configuration. Whereas a configuration can be any arrangement of tiles, we are interested in the subclass of configurations that can result from a self-assembly process. Thus, an **assembly** is a configuration that can result from the start configuration by a sequence of additions of tiles according to the above rules at $\tau = 1$ or $\tau = 2$ (i.e., it is connected). A τ -**stable** assembly is one that cannot be split into two parts without breaking bonds with a total strength of at least τ . **Deterministic** tile systems are those whose assemblies can incorporate at most 1 tile type at any location at any time.

3.2.2 The Kinetic Tile Assembly Model and Errors

The kinetic Tile Assembly Model (kTAM) augments the abstract Tile Assembly Model with a stochastic model of self-assembly dynamics, allowing calculation of error rates and the duration of self-assembly. Following [23, 24] we make the following assumptions. First, the concentration of each tile type in solution is held constant throughout the self-assembly process, and the concentrations of all tile types are equal. We

assume that for every tile association reaction there is a corresponding dissociation reaction (and no others). We further assume that the rate of addition (**forward rate** f) of any tile type at any position of the perimeter of the growing assembly is the same. Specifically, $f = k_f e^{-G_{mc}}$ where k_f is a constant that sets the time scale, and G_{mc} is the logarithm of the concentration of each tile type in solution. The rate that a tile falls off the growing assembly (**reverse rate** r_b) depends exponentially on the number of bonds that must be broken. Specifically, $r_b = k_f e^{-bG_{se}}$ where b is the total interaction strength with which the tile is attached to the assembly, and G_{mc} is the unit bond free energy, which may depend, for example, on temperature.

We assume the following concerning f and r_b . Following [23] we let $f \approx r_2$ for a $\tau = 2$ system since it provides the optimal operating environment [23]. Further, we assume f (and therefore r_2) can be arbitrarily chosen in our model by changing G_{mc} and G_{se} , for example by changing tile concentrations and temperature. (In practice, there are limits to how much these parameters can be changed.) However, k_f is assumed to be a physical constant not under our control.

In the kTAM, the $\tau = 2$ tile addition requirement imposed by the abstract Tile Assembly Model is satisfied only with a certain probability: assuming $f \approx r_2$ so $r_1 \gg f$, if a tile is added that bonds only with strength 1, it falls off very quickly as it should in the aTAM with $\tau = 2$. Tiles attached with strength 2 stick much longer, allowing an opportunity for other tiles to attach to them. Once a tile is bonded with total strength 3, it is very unlikely to dissociate (unless surrounding tiles fall off first).

Following [4], the fundamental kind of error we consider here is an **insufficient attachment**. At threshold $\tau = 2$, an insufficient attachment occurs when a tile attaches with strength 1, but before falling off, another tile attaches next to it, resulting in a 2-stable assembly. Since insufficient attachments are the only kind of error we analyze in this paper, we'll use "error" and "insufficient attachment" interchangeably.

Chen and Goel [4] make use of a simplification of the kTAM that captures the essential behavior while being more tractable for rigorous proofs. Under the conditions where $f = r_2$, the self-assembly process is dominated by tiles being added with exactly 2 bonds and tiles falling off via exactly 2 bonds. The **locking** kTAM model assumes that these are the only possible single-tile events. That is, $r_b = 0$ for $b \geq 3$, and tiles never attach via a single strength-1 bond. Additionally, insufficient attachments are modeled in the locking kTAM as atomic events, in which two tiles are added simultaneously at any position in which an insufficient attachment can occur. Specifically, any particular pair of tile types that can create an insufficient attachment in the kTAM is added at a rate $f_{err} = O(e^{-3G_{se}})$. (This is asymptotically the rate that insufficient attachments occur in kTAM [4].) Thus the total rate of insufficient attachments at a particular location is Qf_{err} , where Q is the number of different ways (with different tile types) that an insufficient attachment can occur there. We don't absorb Q into the $O(\cdot)$ notation because we will be considering tile sets with an increasing number of tile types that can cause errors. Note that Q can be bounded by the square of the total number of tile types. These insufficient attachments are the sole cause of errors during growth.* Growth during which no insufficient attachments occur we call (reversible) $\tau = 2$ **growth**.

3.2.3 Quarter-Plane Patterns

The output of the self-assembly process is usually considered to be either the shape of the uniquely produced terminal assembly [10, 1, 2] (also see Chapter 2) or the pattern produced if we focus on the locations of certain types of tiles [24, 4, 17, 6]. Here we will focus on self-assembling of **quarter-plane patterns**. A quarter-plane pattern (or just **pattern** for short) \mathbf{P} is an assignment of symbols from a finite alphabet of "colors" to points on the quarter plane ($\mathbb{Z}^+ \times \mathbb{Z}^+$ by convention). A deterministic tile system can be thought to construct a pattern in the sense that there is some function (not necessarily a bijection) mapping tile types to colors such that tiles in any produced assembly correctly map to corresponding colors of the pattern. As the assembly grows, a larger and larger portion of the pattern gets filled. There are patterns that cannot be deterministically constructed by any tile system (e.g., uncomputable ones), but for the purposes of this paper

* Another error, with respect to the aTAM, that can occur in the original kTAM is when a tile attached by strength 3 (or more) falls off. Why do we feel comfortable neglecting this error in the locking kTAM, especially since as a function of G_{se} , both r_3 and f_{err} are both $O(e^{-3G_{se}})$? One reason is that in practice the dissociation of tiles held to the assembly with strength 3 does not seem to cause the problems that insufficient attachments induce, in tile sets that we have simulated and examined: no incorrect tiles are immediately introduced, often the correct tile will quickly arrive to repair the hole, and if an incorrect tile fills the hole, further growth may be impossible, usually allowing time for the incorrect tile to fall off. A second reason is that as the number of tile types increases (i.e., with more complex patterns or more complex proofreading schemes), Qf_{err} becomes arbitrarily large, while r_3 stays constant. Nonetheless, a more satisfying treatment would not make these approximations and would address the original kTAM directly.

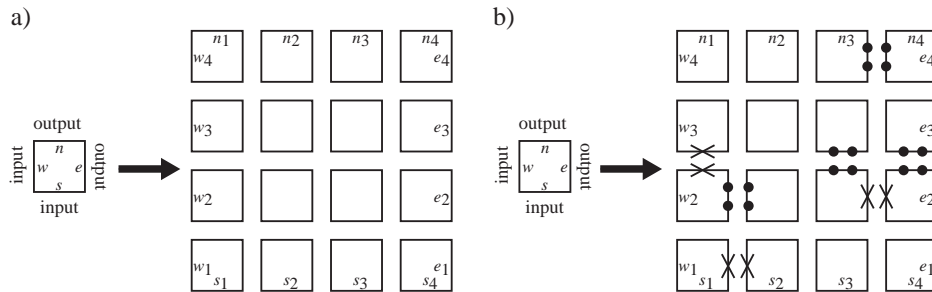


Figure 3.2: Winfree and Bekbolatov (a) and Chen and Goel (b) proofreading transformations using 4×4 blocks. Each tile type is replaced with k^2 tile types that fit together to form the block as shown. Strength 2 bonds are indicated with 2 dots. Strength 0 bonds are indicated with a cross. All unlabeled (internal) bond types are unique (within the block and between blocks.) The placement of weak and strong bonds is dependent upon the orientation of growth, which in this case is to the north-east, since for quarter-plane tile systems the input is always received from the west and south sides.

we consider patterns constructible from deterministic tile systems where all bond strengths are 1 and the seed assembly (defining the boundary conditions) is an infinite L shape that is eventually periodic, with its corner on the origin. See Fig. 3.1 for two examples. Such tile system we'll call **quarter-plane tile systems** and the patterns produced by them the **constructible quarter-plane patterns**. These systems include a wide variety of patterns, including the Sierpinski pattern, the binary counter pattern, the Hadamard pattern [6], and patterns containing the space-time history of arbitrary 1-D block cellular automata and Turing machines. Note that by including the infinite seed assembly we are avoiding the issue of nucleation, which requires distinct error-correcting techniques [18].

3.3 Making Self-Assembly Robust

The kinetic Tile Assembly Model predicts that for any quarter-plane tile system, arbitrarily small error rates can be achieved by increasing G_{mc} and G_{se} , but at the cost of decreasing the overall rate of assembly. Specifically, the worst case analysis (which assumes that after any single error, assembly can be continued by valid $\tau = 2$ growth) predicts that the relationship between per tile error rate ε and the rate of assembly r (layers per second) approximately satisfies $r \propto \varepsilon^2$ [23]. This is rather unsatisfactory since, for example, decreasing the error rate by a factor of 10 necessitates slowing down self-assembly by a factor of 100.

Rather than talking about the relationship between the per tile error rate and the total rate of self-assembly, following [4] one can ask how long it takes to produce the correct $N \times N$ initial portion of the pattern with high probability. To produce this initial portion correctly with high probability, we need the per-tile error rate to be $\varepsilon = O(N^{-2})$ to ensure that no mistake occurs. This implies that $r = O(N^{-4})$ for worst case tile sets. This informal argument suggests that the time to produce the $N \times N$ square is $\Omega(N^4)$. This is unsatisfactory, because the same assembly can be grown in time $O(N)$ in the aTAM augmented with rates [1], and thus the cost of errors appears to be considerable.

Despite this pessimistic argument, certain kinds of tile systems can achieve better error rate/rate of assembly tradeoffs. Indeed, the reversibility of the self-assembly process can help. Some tile systems have the property that upon encountering an error, unless many more mistakes are made, the self-assembly process stalls. Stalling gives time for the incorrectly incorporated tiles to be eventually replaced by the correct ones in a random walk process, so long as not too many incorrect tiles have been added.

Exploiting this observation, several schemes have been proposed for converting arbitrary quarter-plane tile systems into tile systems producing a *scaled-up* version of the same pattern, resulting in better robustness to error. The initial proposal due to Winfree and Bekbolatov [24] suggests replacing each tile type of the original tile system with k^2 tile types, with unique internal strength-1 bonds (Fig. 3.2(a)). Such proofreading assemblies have the property that for a block corresponding to a single tile in the old system to get completed, either no mistakes, or at least k mistakes must occur. However, this scheme suffers from the problem that the

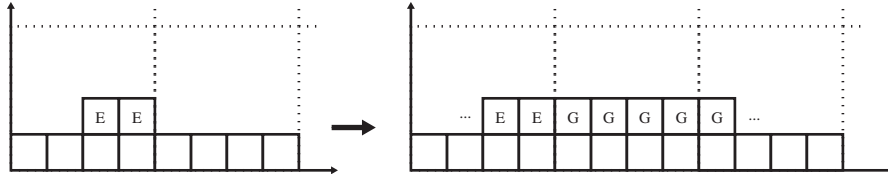


Figure 3.3: The Winfree and Bekbolatov proofreading scheme is susceptible to single facet nucleation errors. If an insufficient attachment results in the two E tiles shown, then subsequent $\tau = 2$ growth (G) can continue indefinitely to the right. Thus many incorrect tiles can be added following a single facet nucleation error even if the block that E is in does not get completed. The dotted lines indicate block boundaries (for 4×4 blocks). Note that most of the incorrect tiles are attached with strength 3; therefore, they do not easily fall off, except at the left and the right sides.

self-assembly process after a single insufficient attachment can still result in a large number of incorrect tiles that must later be removed, spanning the length of the assembly. Consider the situation depicted in Fig. 3.3. If the insufficient attachment illustrated occurs (The first E is added with interaction strength 1, but before it dissociates, a tile attaches to it on the right with interaction strength 2), the incorrect information can be propagated indefinitely to the edge of the assembly by subsequent $\tau = 2$ tile additions.

Currently the only scheme that provably achieves a guaranteed level of proofreading is due to Chen and Goel [4] using the locking kTAM model. Their proofreading scheme, called *snaked proofreading*, is similar to the Winfree and Bekbolatov system, but additionally controls the order of self-assembly within each block by using strength-0 and strength-2 bonds, making sure that not too many incorrect tiles can be added by $\tau = 2$ growth after an insufficient attachment. In particular, the strength-0 bonds ensure that unless most of the block gets completed, self-assembly stalls. Fig. 3.2(b) shows their 4×4 construction; see their paper for the general construction for arbitrary block size.* They can attain a polynomial decrease in the error rate with only a logarithmic increase in k . Specifically the formal results they obtain are the following:^{†‡}

Theorem 3.3.1 (Theorem 4.2 of [4]). *For any constant $p < 1$, the $N \times N$ block initial portion of the pattern is produced correctly with probability at least p in time $O(N \text{ poly}(\log(N)))$ by the $k \times k$ snaked proofreading tile system where $k = \theta(\log N)$, using the locking kTAM with appropriate G_{mc} and G_{se} .*

To obtain this result, assembly conditions (G_{mc} and G_{se}) need be adjusted only slightly as N increases.[§]

The above construction requires increasing the scale of the produced pattern, even if only logarithmically in the size of the total desired size of the self-assembled pattern. Reif et al. [17] pointed this out as a potential problem and proposed schemes for decreasing the effective error rate while preserving the scale of the pattern. However, they rely on certain specific properties of the original tile system, and do not provide a general construction that can be extended to arbitrary levels of error correction. Further, their constructions suffer from the same problem as the original Winfree and Bekbolatov proofreading system. In the next section we argue that the snaked proofreading construction can be adopted to achieve same-scale proofreading for sufficiently “simple” patterns.

*Note that, unlike the original proofreading transformation, the snake proofreading transformation does not result in a quarter-plane tile system as it uses both strong and weak bonds.

[†][4] also guarantees that the assembly is stable for a long time after it is complete, a concern we ignore in this paper. For fixed k , they also provide Theorem 4.1, which guarantees reliable assembly of an $N \times N$ square in time $O(N^{1+8/k})$.

[‡]Chen and Goel only prove their result for the case when the initial L seed assembly has arms that span exactly N blocks. We need to cover the case when an infinite L seed assembly is used. See Appendix 3.6.1 for a proof that their results can be extended to an infinite seed assembly.

[§]It is hard to say whether the snaked proofreading construction is asymptotically optimal. While the best possible assembly time in a model where concentrations are held constant with changing N is linear in N , we assume that G_{mc} and G_{se} are free to change as long as the relationship $f = r_2$ is maintained. Of course while decreasing G_{mc} and G_{se} speeds up the assembly process, the rate of errors is increased; thus, the optimal tradeoff is not obvious.

3.4 Compact Proofreading Schemes for Simple Patterns

In this section we argue that a wide variety of sufficiently “simple” patterns can be produced with arbitrarily small effective error rates without increasing the scale of self-assembly, at the cost of slightly increasing the number of tile types and the time of self-assembly. Based on Reif et al.’s nomenclature [17], we call these proofreading schemes *compact* to indicate that the scale of the pattern is not allowed to change.

The following definition illustrates our goal:

Definition 3.4.1. *Let $p < 1$ be a constant (e.g., 0.99). A sequence of deterministic tile systems $\{\mathbf{T}_1, \mathbf{T}_2, \dots\}$ is a **compact proofreading scheme** for pattern \mathbf{P} if:*

(1: correctness) \mathbf{T}_N produces the full infinite pattern \mathbf{P} under the aTAM.

(2: conciseness) \mathbf{T}_N has $\text{poly}(\log N)$ tile types.

(3: robustness) \mathbf{T}_N produces the correct $N \times N$ initial portion of pattern \mathbf{P} (without scaling) with probability at least p in time $O(N \text{poly}(\log N))$ in the locking kTAM for some G_{se} and G_{mc} .

If you want to construct the initial $N \times N$ portion of pattern \mathbf{P} with probability at least p in time $O(N \text{poly}(\log N))$ you pick tile system \mathbf{T}_N and the corresponding G_{se} and G_{mc} . The same tile system might be used for many N (i.e., the sequence of tile systems may have repetitions). The second condition indicates that we don’t want this tile system to have too many tile types. For constructible quarter-plane patterns, a constant number of tile types suffices to create the infinite pattern in the absence of errors. If the second condition is satisfied then the error correction itself is accomplished with a polylogarithmic number of additional tiles, which is comparable to the cost of error correction in other models studied in computer science. While one can imagine different versions of these conditions, the stated version gives the proofreading condition that can be obtained by adapting the snaked proofreading construction, as argued below. Finally, note that the tile systems $\{\mathbf{T}_1, \mathbf{T}_2, \dots\}$ do not have to be quarter-plane tile systems, and therefore our theorems will apply to a wide range of potential proofreading schemes.

For which patterns do there exist compact proofreading schemes? Given a pattern and a quarter-plane tile system \mathbf{T} producing it, consider any assembly of \mathbf{T} . For a given k , imagine splitting the assembly into $k \times k$ disjoint blocks starting at the origin. We’ll use the term **block** to refer to aligned blocks, and **square** to refer to blocks without the restriction that they be aligned to integer multiples of k with respect to the origin. Each complete block contains k^2 tiles; two blocks at different locations are considered equivalent if they consist of the same arrangement of tile types. If there is some polynomial $Q(k)$ such that repeating this process for all assemblies and all k yields at most $Q(k)$ different (completed) block types, then we say that \mathbf{T} **segments** into $\text{poly}(k)$ $k \times k$ block types.* Patterns produced by such tile systems are the “simple” patterns, for which, we will argue, there exist compact proofreading schemes; we term such patterns **robust** to indicate this.

On the other hand, there are patterns for which it is easy to see that no quarter-plane tile system producing them segments into $\text{poly}(k)$ $k \times k$ block types. For example these include patterns which have $2^{\Omega(k)}$ different types of $k \times k$ squares of colors.† We’ll prove negative results about such patterns, which we term **fragile** in the next section.‡

Definition 3.4.2. *A pattern \mathbf{P} is called **robust** if it is constructible by a quarter-plane tile system \mathbf{T} that segments into $\text{poly}(k)$ different $k \times k$ block types. A pattern \mathbf{P} is called **fragile** if every quarter-plane tile system segments into $2^{\Omega(k)}$ different $k \times k$ block types.*

*We use disjoint blocks aligned with the origin for simplicity in what follows. It is inessential that we define segmentation in terms of blocks rather than squares: A tile system segments into $\text{poly}(k)$ different $k \times k$ block types if and only if it produces assemblies that contain $\text{poly}(k)$ different types of non-aligned $k \times k$ squares. This is also true for other shapes than squares, as long as they have sufficient extent. See Appendix 3.6.2 for an example, the size- k diagonals.

†In what follows, we will consider both the number of blocks (or squares) in an assembly, in which case we mean blocks (or squares) of tile types, as well as the number of blocks (or squares) in a pattern, which which case we mean block (or squares) of colors. Since each tile type has a color, the latter is less than or equal to the former for patterns produced by quarter-plane tile systems.

‡Analogous to the uncomputability of topological entropy for cellular automata [11], it is in general undecidable whether a tile set produces a robust or fragile pattern, due to the undecidability of the Halting Problem: a tile system that simulates a universal Turing machine may either produce a pattern that is eventually periodic (if the Turing machine halts), or else it may continue to produce ever more complicated subpatterns. The former patterns (that are eventually periodic) are formally robust, although only for very large k does this become apparent, while the latter patterns are fragile.

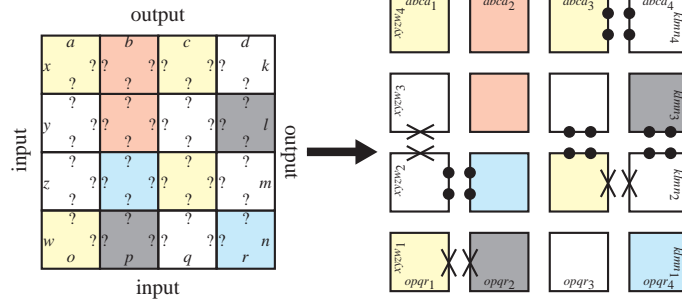


Figure 3.4: Compact proofreading transformations using 4×4 blocks. Strength 2 bonds are indicated with 2 dots. Strength 0 bonds are indicated with a cross. Question marks indicate arbitrary bond types. All unlabeled (internal) bond types are unique (within the block and between blocks.) This construction is equivalent to “compressing” the $k \times k$ block on the left to a single tile and then applying the snaked proofreading construction, remembering to paint the resulting tiles with the original colors.

The natural way to use Chen and Goel’s construction to implement compact proofreading for robust patterns is as follows. For any k , for each of the $\text{poly}(k)$ $k \times k$ block types described above, create k^2 unique tile types with bond strengths according to the snaked proofreading blocks and colors according to the original pattern. The internal bond types are unique to each transformed $k \times k$ block type and do not depend upon the internal bond types in the original $k \times k$ block type. External bond types in the transformed block redundantly encode the full tuple of external bond types in the original block. (This transformation for a 4×4 block is illustrated in Fig. 3.4.) The L-shaped seed assembly must also be revised to use the new compound bond types. The above set of tile types together with this seed assembly yields a new tile system $\mathbf{T}(k)$.

It is easy to check that under aTAM $\mathbf{T}(k)$ correctly produces the pattern. At the corner between two existing blocks, only a tile that matches all the border tiles of both blocks can attach. Any other internal tile must bind correctly since at least one side must match a bond type unique to the block. Since the original block assembled deterministically from its west and south sides, the transformed block also grows deterministically in the same direction. In fact, $\mathbf{T}(k)$ is locally deterministic [20], which makes a formal proof easy. Furthermore, for any particular choice of k , Chen and Goel’s Theorem 4.1 applies directly to our compact proofreading tile sets, but with multiplicative constants that increase with k . But we also claim the following, where $M = \lceil N/k \rceil$ is the size of our target assembly in units of blocks:

Lemma 3.4.1. *If a pattern \mathbf{P} is robust then: For any constant $p < 1$, the $M \times M$ block initial portion of the pattern is produced correctly with probability at least p in time $O(M \text{poly}(\log M))$ by some $\mathbf{T}(k)$ (as defined above) where $k = \theta(\log M)$, using the locking k TAM with appropriate G_{mc} and G_{se} .*

Proof. Recall, as long as a particular location remains susceptible, insufficient attachments at that location constitute a Poisson process with rate $QO(e^{-3G_{se}})$. Here Q can be upper bounded by the total number of different blocks since that is the maximum number of different tile types that can be added as an insufficient attachment at any location. Thus, the maximum rate of insufficient attachments at any location is $q(G_{se}) = Q(k)O(e^{-3G_{se}})$, where $Q(k) = \text{poly}(k)$ since the pattern is robust.

The difference between the proof of Chen and Goel [4] and what we need is that Chen and Goel assumed that $Q(k)$ was a constant. Thus, whereas they were able to increase k without increasing the rate of insufficient attachments, q , we are not so fortunate. To remedy this situation, we must slow down growth slightly in order to sufficiently decrease the rate of insufficient attachments, but not so fast as to change the asymptotic form of the results.

Informally, note that Chen and Goel’s bound on the probability of successfully completing the square within a certain time (scaled relative to f) depends only on the ratio q/f ; the absolute time scale does not matter, nor does it matter whether q is the result of many or a few possible erroneous block types. Thus, we can slow down f by a polynomial in k without affecting the completion time asymptotics of $O(M \text{poly}(\log M))$, since $k = \Theta(\log M)$. Does q decrease enough? So long as it decreases faster relative to f , we can compensate for the polynomial increase in insufficient attachments. We will see that a factor of $Q(k)^2$ is sufficient.

Formally, assuming the maximum rate of insufficient attachments is any $\tilde{q}(G_{se}) = O(e^{-3G_{se}})$ independent of k , and the forward (=reverse) rate is any $\tilde{f}(G_{se}) = \Omega(e^{-2G_{se}})$, for any M , Chen and Goel give a value \tilde{k} for k and \tilde{G}_{se} for G_{se} such that with high probability the assembly completes correctly in time $t = O(M \text{ poly}(\log M))$. We, of course, have $q(G_{se}) = O(Q(k)e^{-3G_{se}})$ and $f(G_{se}) = \Omega(e^{-2G_{se}})$. Now let us define $\tilde{q}(G_{se}) = q(G_{se} + \ln Q(k)) \cdot Q(k)^2$ and $\tilde{f}(G_{se}) = f(G_{se} + \ln Q(k)) \cdot Q(k)^2$. Observe that $\tilde{q}(G_{se}) = O(e^{-3G_{se}})$ and $\tilde{f}(G_{se}) = O(e^{-2G_{se}})$. This means that if the maximum rate of insufficient attachments and the forward rate were these \tilde{q} and \tilde{f} , then Chen and Goel's proof gives values \tilde{k} and \tilde{G}_{se} such that with high probability the assembly completes correctly in time $t = O(M \text{ poly}(\log M))$. But now note that if we set $G_{se} = \tilde{G}_{se} + \ln Q(\tilde{k})$, then the actual maximum rate of insufficient attachments and the forward rate are both exactly a factor of $Q(\tilde{k})^2$ slower than \tilde{q} and \tilde{f} . Thus our system is simply overall slower by a factor of $Q(\tilde{k})^2$. This means that our system would finish correctly with the same high probability as achieved by Chen and Goel by time $O(tQ(\tilde{k})^2)$. But this is still $O(M \text{ poly}(\log M))$ since $\tilde{k} = \theta(\log M)$ and $Q(\tilde{k}) = \text{poly}(\tilde{k})$. \square

Theorem 3.4.1. *If a pattern \mathbf{P} is robust then there exists a compact proofreading scheme for \mathbf{P} .*

Proof. Let us use the sequence $\{\mathbf{T}_N = \mathbf{T}(k)\}_N$ where k for each N is from Lemma 3.4.1. Each of these tile systems can produce the whole pattern correctly under aTAM so the correctness condition of Definition 3.4.1 is satisfied. Since $O(M \text{ poly}(\log M)) = O(N \text{ poly}(\log N))$, Lemma 3.4.1 implies that the sequence satisfies the robustness condition. Further, because \mathbf{T} segments into $\text{poly}(k)$ different $k \times k$ block types and $k = \theta(\log M)$ implies $k = O(\log N)$, $\mathbf{T}_N = \mathbf{T}(k)$ has only $\text{poly}(k)k^2 = \text{poly}(\log N)$ tile types, satisfying the conciseness condition. \square

For some patterns, Chen and Goel's theorem can be applied directly (without requiring Lemma 3.4.1). These include patterns whose quarter-plane tile systems segment into a constant number of $k \times k$ block types. Furthermore, consider the Sierpinski pattern (Fig. 3.1(c)). The Sierpinski pattern is a fractal that has the following property: split the pattern into blocks of size $k \times k$ for any k that is a power of 2, starting at the origin. For any such k there are exactly 2 different types of blocks in the pattern. If you consider the assembly produced by the Sierpinski tile system in Fig. 3.1(d), there are exactly 4 different $k \times k$ blocks of tiles (the difference is due to the fact there are now two types of black and two types of white tiles.) We can let the sequence of tile systems for the compact proofreading scheme for the Sierpinski pattern consist only of $\mathbf{T}(k)$ for k that are a power of 2. Note that because of the restriction on k , we may have to use a block size larger than that which results from Chen and Goel's theorem. But since it does not have to be more than twice as large, Definition 3.4.1 is still satisfied.

It would be interesting to identify constructible quarter-plane patterns that have at least k^d different $k \times k$ block types for all k and for some constant $d \geq 1$.

3.5 A Lower Bound

In this section we will show that we cannot make compact proofreading schemes for fragile patterns using known methods.

First of all, note that although the definition of fragile patterns quantifies over all quarter-plane tile systems, it can be very easy to prove that a pattern is fragile using the following lemma.

Lemma 3.5.1. *If a pattern \mathbf{P} has $2^{\Omega(k)}$ different types of $k \times k$ squares of colors then it is fragile.*

Proof. If a pattern contains $2^{\Omega(k)}$ different types of $k \times k$ squares of colors, then any tile system producing it contains at least $2^{\Omega(k)}$ different types of $k \times k$ squares, and therefore comparably many block types. \square

The scheme described in the previous section does not work for quarter-plane tile systems that segment into $2^{\Omega(k)}$ $k \times k$ block types (i.e., fragile patterns). This is because for $k = \theta(\log N)$, $\mathbf{T}(k)$ would then have $\text{poly}(N)$ tile types, violating the second condition (conciseness) of compact proofreading schemes (Definition 3.4.1).^{*} However, it is unclear whether other methods exist to make compact proofreading schemes for

^{*}Further, we believe Lemma 3.4.1 does not hold if the number of block types increases exponentially, rather than polynomially in k . This is an open question.

patterns produced by such tile systems. While we cannot eliminate this possibility entirely, we can show that a variety of schemes will not work.

Existing attempts at making self-assembly robust through combinatorial means ([24, 4, 17]) are based on creating redundancy in the produced assembly. Specifically, knowing only a few tiles allows one to figure out a lot more of the surrounding tiles. Intuitively, this redundancy allows the tile system to “detect” when an incorrect tile has been incorporated and stall. We will argue that if a pattern is sufficiently complex, then only if there are many possible tile types can a few tiles uniquely determine a large portion of the pattern. Since the definition of compact proofreading schemes (Definition 3.4.1) limits the number of tile types, we will be able to argue that for complex patterns there do not exist compact proofreading schemes that rely on this type of redundancy.

Definition 3.5.1. *An assembly A is (k, d) -redundant if there exists a decision procedure that, for any $k \times k$ (completed) square of tiles in A , querying at most d relative locations in the assembly for its tile type, can determine the types of all tiles in that square.*

The proofreading schemes of [24] and [4], using a block size $k \times k$, are $(k, 3)$ -redundant: even if the square is not aligned with the blocks, it is enough to ask for the types of the tiles in the upper-left, lower-left, and lower-right corners of the square. Because all tiles in a block are unique, and because the tile system is deterministic, these three tiles allow you to figure out all four blocks that the square may intersect. A proofreading construction that generalizes Reif et al.’s [17] 2-way and 3-way overlay tile sets to k -way overlays is shown in Appendix 3.6.2 to be $(k, 3)$ -redundant as well. This construction is not based on block transformations; the fact that its power is nonetheless limited by Theorem 3.5.1, below, illustrates the strength of our lower bound.

Lemma 3.5.2. *If a tile system \mathbf{T} produces (k, d) -redundant assemblies in which more than 2^{ck} different types of (completed) $k \times k$ squares appear, then it must have at least $2^{ck/d}$ tile types.*

Proof. Let m be the number of tile types of \mathbf{T} . If an assembly produced by \mathbf{T} is (k, d) -redundant, then it has no more than m^d types of squares of size $k \times k$ because the decision procedure’s decision tree is of depth at most d and of fan-out at most m . But we assumed that \mathbf{T} makes assemblies that have 2^{ck} different types of $k \times k$ squares. Thus, $m^d \geq 2^{ck}$, which can only happen if $m \geq 2^{ck/d}$. \square

Lemma 3.5 lets us limit the types of compact proofreading schemes that such complex patterns may have.

Theorem 3.5.1. *If a pattern is fragile then there does not exist a compact proofreading scheme $\{\mathbf{T}_1, \mathbf{T}_2, \dots\}$ such that \mathbf{T}_N produces assemblies that are $(\Omega(\log N), d)$ -redundant (for any constant d).*

Proof. Any tile system producing this pattern makes $2^{\Omega(k)}$ different types of $k \times k$ (completed) squares of tiles. Suppose \mathbf{T}_N produces assemblies which are $(c' \log N, d)$ -redundant, for constants c', d . Take $k = c' \log N$ and note that for large k , \mathbf{T}_N makes at least 2^{ck} $k \times k$ squares for some constant c . Apply Lemma 3.5.2 to conclude that \mathbf{T}_N has at least $2^{ck/d} = N^{cc'/d}$ tile types, which violates the second condition of Definition 3.4.1. \square

Even though both the Sierpinski pattern and the counter pattern (Fig. 3.1) are infinite binary patterns that can be constructed by very similar tile systems, they are very different with respect to error correction. We saw that the Sierpinski pattern has compact proofreading schemes. However, because the counter must count through every binary number, for any k there are 2^k rows that have different initial patterns of black and white squares. This implies that there are exponentially many (in k) different squares. By Theorem 3.5.1 this implies that the counter pattern does not have compact proofreading schemes that use $(\Omega(\log N), d)$ -redundant assemblies. That is, no existing proofreading scheme can be adapted for making compact binary counters arbitrarily reliable.

This theorem suggests that in order to find universal compact proofreading schemes we must find a method of making self-assembly more error-robust without making it too redundant. However, we conjecture that there are inherent trade-offs between robustness and conciseness (small number of tile types), raising the possibility that there do not exist compact proofreading schemes for patterns having an exponential number of $k \times k$ squares.

3.6 Appendix

3.6.1 Extension of Chen and Goel’s Theorem to Infinite Seed Boundary Assemblies

The following argument uses terms and concepts from [4].

First, suppose we desire to build an $(N + N^2) \times (N + N^2)$ block initial portion of the pattern starting with the L seed assembly having arms that are $N + N^2$ blocks long. The extra N^2 blocks will serve as a buffer region. Chen and Goel’s [4] Theorem 4.2 then gives us a $k = \theta(\log(N + N^2)) = \theta(\log N)$ and G_{se} s.t. with high probability no block error occurs in the $(N + N^2) \times (N + N^2)$ block region in time $O(N^2 \text{poly}(\log N))$ that it takes to finish it. Further, with high probability the initial $N \times N$ block portion of the pattern is completed in time $t_N = O(N \text{poly}(\log N))$.

Now, let’s suppose we use this k and G_{se} with an infinite L seed assembly, and we’ll be interested in just the $N \times N$ block initial portion of the pattern. The only way the infinite seed assembly can affect us is if a block error outside the $(N + N^2) \times (N + N^2)$ block region propagates to the $N \times N$ initial region before it completes. For this to occur, at least N^2 tiles must be added sequentially, at least one per block through the buffer region, to propagate the error. The expected time for this to happen is N^2/f with standard deviation N/f (i.e., it is a gamma distribution with shape parameter N^2 and rate parameter f). However, the propagated error can only cause a problem if it reaches the $N \times N$ rectangle before time t_N . Since $t_N = O(N \text{poly}(\log N))$, this becomes less and less likely as N increases by Chebyshev’s inequality. Small N are handled by increasing k and G_{se} appropriately, which does not affect the asymptotic results. Thus we have a $k = \theta(\log N)$ and G_{se} such that with high probability (i.e., $\geq p$) the initial $N \times N$ block portion of the pattern is completed correctly in time $O(N \text{poly}(\log N))$, even if we use an infinite L seed assembly.

3.6.2 An Overlay Proofreading Scheme

In this appendix we give an example showing that our lower bound on the complexity of same-scale proofreading schemes also applies to proofreading schemes that are not based on block transformations. Here, we consider a k -way overlay scheme (suggested by Paul Rothemund and Matt Cook) that generalizes the 2-way and 3-way overlay schemes introduced by Reif et al. [17]. The construction is shown in Fig. 3.5.

Consider the assembly grown using some original tile set, as in Fig. 3.5a. When the shaded tile x was added, it attached to the tiles a and b to its west and to its south. Since we consider only deterministic quarter-plane tile sets, the tile type at a particular location is a function of the tile types to its south and to its west, e.g., $x = f(a, b) = f_{ab}$. Therefore, it is possible to reconstruct the same pattern without keeping track of bond types, explicitly transmitting only information about tile types.

The 1-overlay tile set, derived from the original tile set, is a deterministic tile set for doing exactly that. As shown in Fig. 3.5b, for each triple of neighboring tiles a, b , and x that appears in the assembly produced by the original tile set (in the relative positions shown in (a)), create a new tile (x, x, b, a) , colored the same as x , that “inputs” the original tile types of its west and its south neighbors, and “outputs” tile type x to both its north and its east neighbor. With an appropriately re-coded L-shaped boundary, the new tile set will produce exactly the same pattern as the original tile set: the output of the tile at location $\langle i, j \rangle$ in the 1-overlay assembly is the tile type at $\langle i, j \rangle$ in the original assembly. Supposing the original tile set T has $|T|$ tile types, the new tile set contains at most $|T|^2$ tile types, and possibly fewer if not all pairs of inputs a, b appear in the pattern.

Redundancy is achieved in a k -way overlay tile set by encoding not just one original tile, but k adjacent tiles along the diagonal growth front. Specifically, each tile in the k -way overlay assembly will output the k -tuple of original tile types that appear in the same location in the original assembly and locations to the east and south. For example, in Fig. 3.5c, the output of the tile at $\langle i, j \rangle$ in the 4-overlay assembly is the 4-tuple $abcd$ containing the tile types at locations $\langle i, j \rangle$, $\langle i + 1, j - 1 \rangle$, $\langle i + 2, j - 2 \rangle$, and $\langle i + 3, j - 3 \rangle$. Each new tile is colored according to the first tile type in its output tuple. The new tile set consists of all such tiles that appear in the k -overlay assembly*[†]. The new tile set contains at most $|T|^{k+1}$ tiles, since there are at most

*In addition, the L-shaped boundary must be properly re-coded to carry the boundary information in the form the new tiles require. This is easy to do if the pattern is consistent with a larger hypothetical assembly that extends k tiles beyond the quarter-plane region, since then tuples on the boundary encode for tile types in this buffer zone. Otherwise a few extra tile types will be necessary, but as this does not change the nature of our arguments, we ignore this detail here.

[†]Note that the exact (minimal) set of such tiles is in general uncomputable, since the original tile set could be Turing universal, and

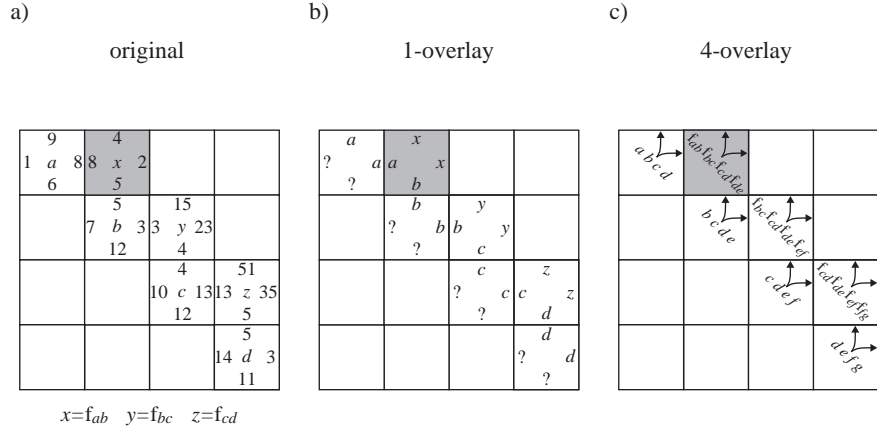


Figure 3.5: The construction for k -way overlay proofreading tile sets. **(a)** An original quarter-plane tile set T , containing $|T|$ tile types. Numbers indicate bond types. Letters name the tile types. For example, the tile $x = (4, 8, 5, 2)$. **(b)** The 1-overlay transformation of the original tile set. The question marks indicate that there may be several different new tile types that output a or b ; **(c)** The 4-overlay transformation of the original tile set

$|T|^k$ input k -tuples, and the two inputs to a given tile will always agree at $k - 1$ indices. This is exponential in k , but for some patterns — e.g., robust patterns, as we will see — only a polynomial number of tile types will be necessary. Note that growth with the new tile set is still deterministic, since the tuple output by a tile is a function of the two input tuples.

In what sense is the k -overlay tile set guaranteed to be proofreading? Consider a growth site where a tile is about to be added. Unless the two input k -tuples agree at all $k - 1$ overlapping positions, there will be no tile that matches both inputs. Thus, every time that a tile is added without a mismatch, it provides a guarantee that $k - 1$ parallel computations are carrying the same information, locally. Note that the fact that site $\langle i, j \rangle$ in the original assembly contains tile type t is encoded in k locations in the k -overlay assembly. It is reasonable to conjecture that it is impossible for all k locations to have incorrect information, unless at least k insufficient attachments have occurred.

Unfortunately, like the original proofreading tile sets of [24] and the 2-way and 3-way overlay tile sets described in [17], the k -way overlay tile sets do not protect against facet nucleation errors, and therefore we do not expect error rates to decrease substantially with k . We do not see an obvious way to correct this deficiency.

Nonetheless, as a demonstration of the general applicability of our lower bound, we will show that *even if* the k -way overlay tile sets reduced errors sufficiently, for fragile patterns the k -way overlay tile sets will contain an exponential number of tile types and are thus infeasible, whereas for robust patterns the k -way overlay tile sets will contain a polynomial number of tile types and are thus feasible.

First we show that all k -overlay tile sets are $(k, 3)$ -redundant, regardless of the original tile set. To determine all tile types in the $k \times k$ square with lower left coordinate $\langle i, j \rangle$, we need only know the tiles at $\langle i, j - 1 \rangle$, $\langle i - k, j + k - 1 \rangle$, and $\langle i + k - 1, j - k \rangle$. The outputs of these tiles encode for the entire diagonal from $\langle i - k, j + k - 1 \rangle$ to $\langle i + 2k - 2, j - 2k + 1 \rangle$ in the original assembly. Deterministic growth from this diagonal results in a triangle of tiles with upper right corner at $\langle i + 2k - 1, j + k - 1 \rangle$, in the original assembly. Thus all tile types are known for the input and output k -tuples of overlay tiles in the $k \times k$ square of interest.

Theorem 3.5.1 tells us that fragile patterns cannot have compact proofreading schemes that are $(\Omega(\log N), d)$ -redundant for any constant d . Therefore, k -overlay tile sets can't work as compact proofreading schemes for fragile patterns; they must have an exponential number of tile types. This is what we wanted to show.

Alternatively, we could have directly bounded the number of tile types in k -overlay tile sets for fragile and

thus predicting whether a particular original tile appears in the assembly is equivalent to the Halting Problem. However, the new tile set is well defined and in many cases can be easily computed.

robust patterns. For robust patterns, with $\text{poly}(k)$ $k \times k$ squares of tile types, clearly there are also $\text{poly}(k)$ size- k diagonals. Since each tile in the k -overlay tile set contains two inputs encoding size- k diagonals, there can be at most $\text{poly}(k)^2 = \text{poly}(k)$ tile types altogether. Thus, (although probably not satisfying the robustness criterion of Definition 3.4.1) k -overlay tile sets are at least concise for robust patterns. Conversely, concise k -overlay tile sets, having $\text{poly}(k)$ tile types by construction, have a comparable number of size- k diagonals in the original assembly. Consider now the original assembly. Since growth is deterministic, the diagonal determines the upper right half of a $k \times k$ square, and thus there are $\text{poly}(k)$ tops and $\text{poly}(k)$ sides; taking these as inputs to other squares, we see that there are $\text{poly}(k)^2 = \text{poly}(k)$ $k \times k$ squares. In this loose sense, k -overlay tile sets are neither more nor less concise than $k \times k$ snaked proofreading, for robust patterns.

On the other hand, for a fragile pattern, requiring $2^{\Omega(k)}$ $k \times k$ squares of tiles in any tile system that produces it, we can see that there will also be at least $2^{\Omega(k)}$ size- k diagonals of tiles. Specifically, if $S(k)$ is the number of such squares, and $D(k)$ is the number of such diagonals, then $S(k) \leq D(2k)$ because deterministic growth from a size- $2k$ diagonal results in the completion of a triangular region containing a $k \times k$ square. $S(k)$ being at least exponential therefore implies the same for $D(k)$. Conversely, a pattern generated by a tile system with $2^{\Omega(k)}$ size- k diagonals obviously also has at least that many $k \times k$ squares as well. Thus, our notions of fragile and robust patterns appears to be sufficiently general.

Acknowledgments

We thank Ho-Lin Chen, Ashish Goel, Paul Rothmund, Matthew Cook, and Nataša Jonoska for discussions that greatly contributed to this work. This work was supported by NSF NANO Grant No. 0432193.

Bibliography

- [1] L. M. Adleman, Q. Cheng, A. Goel, and M.-D. A. Huang. Running time and program size for self-assembled squares. In *ACM Symposium on Theory of Computing (STOC)*, pages 740–748, 2001.
- [2] G. Aggarwal, M. Goldwasser, M. Kao, and R. T. Schweller. Complexities for generalized models of self-assembly. In *Symposium on Discrete Algorithms (SODA)*, pages 880–889, 2004.
- [3] R. D. Barish, P. W. K. Rothmund, and E. Winfree. Two computational primitives for algorithmic self-assembly: Copying and counting. *NanoLetters*, (to appear).
- [4] H.-L. Chen and A. Goel. Error free self-assembly using error prone tiles. In Ferretti et al. [7], pages 62–75.
- [5] J. Chen and J. Reif, editors. *DNA Computing 9*, volume LNCS 2943. Springer-Verlag, 2004.
- [6] M. Cook, P. W. K. Rothmund, and E. Winfree. Self-assembled circuit patterns. In Chen and Reif [5], pages 91–107.
- [7] C. Ferretti, G. Mauri, and C. Zandron, editors. *DNA Computing 10*, volume LNCS 3384. Springer-Verlag, 2005.
- [8] P. W. K. Rothmund. *Theory and Experiments in Algorithmic Self-Assembly*. PhD thesis, University of Southern California, 2001.
- [9] P. W. K. Rothmund, N. Papakakis, and E. Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology*, 2:e424, 2004.
- [10] P. W. K. Rothmund and E. Winfree. The program-size complexity of self-assembled squares. In *ACM Symposium on Theory of Computing (STOC)*, pages 459–468, 2000.
- [11] L. Hurd, J. Kari, and K. Culik. The topological entropy of cellular automata is uncomputable. *Ergodic Theory Dynamical Systems*, 12:255–265, 1992.

- [12] T. H. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J. H. Reif, and N. C. Seeman. Construction, analysis, ligation, and self-assembly of DNA triple crossover complexes. *Journal of the American Chemical Society*, 122:1848–1860, 2000.
- [13] M. G. Lagoudakis and T. H. LaBean. 2-D DNA self-assembly for satisfiability. In E. Winfree and D. K. Gifford, editors, *DNA Based Computers V*, volume 54 of *DIMACS*, pages 141–154. American Mathematical Society, 2000.
- [14] C. Mao, T. H. LaBean, J. H. Reif, and N. C. Seeman. Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. *Nature*, 407:493–496, 2000.
- [15] C. Mao, W. Sun, and N. C. Seeman. Designed two-dimensional DNA holliday junction arrays visualized by atomic force microscopy. *Journal of the American Chemical Society*, 121:5437–5443, 1999.
- [16] J. Reif. Local parallel biomolecular computing. In H. Rubin and D. H. Wood, editors, *DNA Based Computers III*, volume 48 of *DIMACS*, pages 217–254. American Mathematical Society, 1999.
- [17] J. H. Reif, S. Sahu, and P. Yin. Compact error-resilient computational DNA tiling assemblies. In Ferretti et al. [7], pages 293–307.
- [18] R. Schulman and E. Winfree. Programmable control of nucleation for algorithmic self-assembly. In Ferretti et al. [7], pages 319–328.
- [19] R. Schulman and E. Winfree. Self-replication and evolution of DNA crystals. In M. S. Capcarrere, A. A. Freitas, P. J. Bentley, C. G. Johnson, and J. Timmis, editors, *Advances in Artificial Life: 8th European Conference (ECAL)*, volume LNCS 3630, pages 734–743. Springer-Verlag, 2005.
- [20] D. Soloveichik and E. Winfree. Complexity of self-assembled shapes, 2005. Extended abstract; preprint of the full paper is [cs.CC/0412096](https://arxiv.org/abs/cs/0412096) on arXiv.org.
- [21] E. Winfree. On the computational power of DNA annealing and ligation. In R. J. Lipton and E. B. Baum, editors, *DNA Based Computers*, volume 27 of *DIMACS*, pages 199–221. American Mathematical Society, 1996.
- [22] E. Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, 1998.
- [23] E. Winfree. Simulations of computing by self-assembly. Technical report, Caltech, 1998.
- [24] E. Winfree and R. Bekbolatov. Proofreading tile sets: Error-correction for algorithmic self-assembly. In Chen and Reif [5], pages 126–144.
- [25] E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman. Design and self-assembly of two dimensional DNA crystals. *Nature*, 394:539–544, 1998.
- [26] E. Winfree, X. Yang, and N. C. Seeman. Universal computation via self-assembly of DNA: Some theory and experiments. In L. F. Landweber and E. B. Baum, editors, *DNA Based Computers II*, volume 44 of *DIMACS*, pages 191–213. American Mathematical Society, 1998.

Chapter 4

Combining Self-Healing and Proofreading in Self-Assembly

Collaborators: Matthew Cook and Erik Winfree. **My contribution:** I invented the construction with some discussion with MC. I developed the proofs and wrote the text of the paper.

This chapter was published as: David Soloveichik, Matthew Cook, Erik Winfree, “Combining Self-Healing and Proofreading in Self-Assembly,” *Natural Computing*, (on-line July 2007).

4.1 Abstract

Molecular self-assembly is a promising approach to bottom-up fabrication of complex structures. A major impediment to the practical use of self-assembly to create complex structures is the high rate of error under existing experimental conditions. Recent theoretical work on algorithmic self-assembly has shown that under a realistic model of tile addition and detachment, error-correcting tile sets are possible that can recover from the attachment of incorrect tiles during the assembly process. An orthogonal type of error correction was recently considered as well: whether damage to a completed structure can be repaired. It was shown that such self-healing tile sets are possible. However, these tile sets are not robust to the incorporation of incorrect tiles. It remained an open question whether it is possible to create tile sets that can simultaneously resist wholesale removal of tiles and the incorporation of incorrect ones. Here we present a method for converting a tile set producing a pattern on the quarter-plane into a tile set that makes the same pattern (at a larger scale) but is able to withstand both of these types of errors.

4.2 Introduction

The Tile Assembly Model [21, 22] formalizes a generalized crystal growth process by which an organized structure can spontaneously form from simple parts. It provides the foundation for theoretically examining how to use self-assembly for massively parallel DNA computation [20, 26, 15, 12], for creating objects with programmable morphogenesis [10, 1, 2] (also see Chapter 2), for patterning of components during nanofabrication of molecular electronic circuits [6], and for studying self-replication and Darwinian evolution of information-bearing crystals [16, 17]. In addition to this theoretical work, several self-assembling systems have been implemented experimentally using DNA molecules as tiles, including both periodic [25, 14, 11] and algorithmic patterns [13, 9, 3].

The Tile Assembly Model considers the growth of two dimensional “crystals” made out of square units called tiles. Typically, there are many types of tiles that must compete to bind to the crystal. A new tile can be added to a growing complex if it binds strongly enough. Each of the four sides of a tile has an associated bond type that interacts with abutting sides of other tiles that have already been incorporated. If the two abutting sides have different bond types then their interaction strength is 0. Otherwise, the bond type determines the interaction strength. For tile systems shown in this paper, at least a single strong bond (strength 2) or two

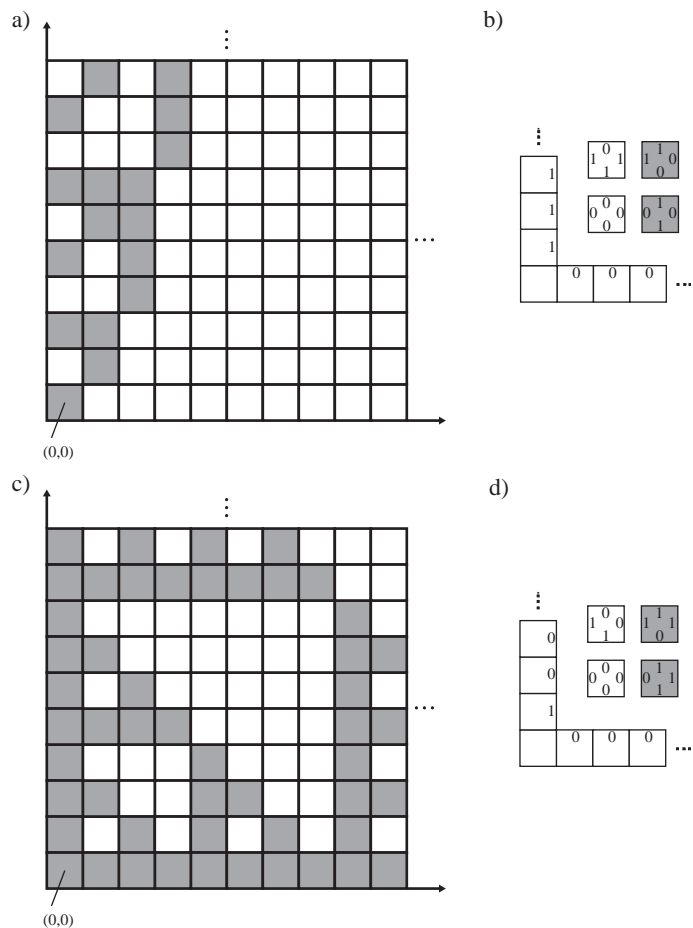


Figure 4.1: (a) A binary counter pattern and (b) a tile system constructing it. (c) A Sierpinski pattern and (d) a tile system constructing it. The L-shaped boundary (represented in (a) and (b) as the x and y axes) is the seed. We assume it is exactly as large as the portion of the pattern we are trying to build. In this formalism, identically-labeled sides match and tiles cannot be rotated. All bond types are weak (strength 1); thus, tiles may attach to the growing assembly only if at least two sides match. Note that the tile choice at each site is deterministic for these two tile sets if the assembly is growing north-east. Growth in the south-west and north-west directions is not deterministic for the counter, and south-west growth is not deterministic for the Sierpinski.

weak bonds (strength 1 each) need to be formed for a tile to attach. This is called *error-free* tile addition. The assembly process starts from a specified seed assembly and proceeds by sequential addition of tiles. An *assembly* is an arrangement of tiles that can result by this process. Tiles do not get used up since it is assumed there is an unbounded supply of tiles of each type. If every tile type is “colored” a certain way, then the self-assembly process can produce a pattern. Fig. 4.1 illustrates two different patterns and the corresponding tile systems that self-assemble into them. Patterns, like these, that grow from an L-shaped boundary are called *quarter-plane* patterns; while more complex growth paths are possible within the Tile Assembly Model, we do not consider them here, because quarter-plane patterns are a rich class (including universal computation) and we feel that their study is sufficient for identifying the essential principles.

A major stumbling block to making algorithmic self-assembly practical is the error rate inherent in current implementations. While the abstract model supposes that tile additions are error-free and permanent, in reality tile additions are error prone and tiles can dissociate from a growing complex. Further, huge chunks of the structure may be physically ripped off by external mechanical forces, such as shear due to fluid flow during sample handling. Erroneous addition of tiles and wholesale removal of tiles have been examined separately

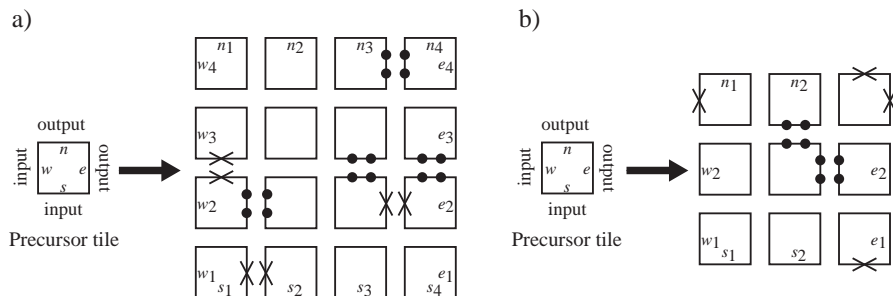


Figure 4.2: (a) Chen and Goel’s snaked proofreading transformations using 4×4 blocks (i.e., $k = 4$), and (b) Winfree’s self-healing transformations for quarter-plane tile systems. Each tile type is replaced with the tile types that fit together to form the block as shown. Strong bonds (strength 2) are indicated with 2 dots. Null bonds (strength 0) bonds are indicated with a cross. All unlabeled (internal) bond types are unique (within the block and between blocks.) The placement of weak and strong bonds is dependent upon the orientation of growth, which is to the north-east for quarter-plane tile systems.

in the literature, so let us review them in turn.

Recent experimental demonstrations of algorithmic self-assembly exhibit error rates of 1% to 15%: on average every eighth to hundredth tile that is incorporated does not correctly bond with its neighbors [9, 3]. Once such a mistake occurs, the erroneous information can be propagated to tiles that are subsequently attached. Thus, a single mistake can result in a drastically different pattern being produced. With this error rate, structures of size larger than roughly 100 tiles cannot be assembled reliably.

While the physics of the self-assembly process could possibly be modified to achieve a lower probability of the incorporation of incorrect tiles into the growing complex, it is also possible to use some logical properties of the tiles to perform error correction [24]. In this vein, Chen and Goel [4] developed *snaked proofreading* tile sets that make use of redundant encoding of information to achieve robustness to error (see Fig. 4.2(a)). Each tile type of the original tile system is replaced by k^2 tile types that form a block corresponding to the original tile and is colored the same. If growth occurs without error, the same pattern is produced, albeit at a k times larger scale. However, an error leads to an assembly whose growth cannot be continued without further errors. Since further errors are unlikely to happen in just the right time and place, growth around erroneous tiles stalls and the erroneous tiles are able to subsequently dissociate, allowing another chance for correct growth. Using this approach, a large class of tile sets can be transformed into more robust tile sets that assemble according to the same logic at a larger scale k . Chen and Goel were able to prove, with respect to a reversible model of algorithmic self-assembly, that error rates decrease exponentially with k , and thus making an $N \times N$ initial portion of the pattern, which requires an error rate of $\approx 1/N^2$, can be done in time $O(N \text{poly}(\log(N)))$ using only $k = \Omega(\log N)$.

Extensive damage to the completed parts of the structure was considered in [23]. Damage caused by external destructive physical processes is modeled by simply removing some number of tiles from the growing (or completed) structure. Because the assembly model allows crystals to grow in any direction, tiles may begin to fill in holes in the structure from a different direction than the direction of their original growth. While forward growth was deterministic, most of the time backward and sideways growth is not (unless the computation being performed is reversible in some sense). For example, both the binary counter and the Sierpinski pattern do not have deterministic backward growth. *Self-healing* tile sets were developed that perfectly heal such damage to the self-assembled object, assuming that only error-free tile additions occur (see Fig. 4.2(b)). Each tile in the original tile set is replaced with 9 tiles as shown in the figure, and thus the pattern is produced at a fixed scale-up factor of 3.* The key idea of the construction is that it guarantees that the regrowth occurs from the same direction as original growth by the placement of null bonds (strength 0) that prevent backward growth and strong bonds (strength 2) that allow the assembly process to proceed correctly in the forward direction.

*Allowing self-assembly to start from a preexisting seed boundary as in this paper, rather than from a single seed tile as in [23], actually permits the use of a simpler transformation that produces a scale-up factor of just 2.

In summary we have two types of errors: (1) tile additions that violate the rule that a tile may only be added if it binds strongly enough, and (2) the removal of tiles despite them being strongly bonded. With existing techniques, each of these types of errors can be controlled separately, but not when they can occur in the same assembly process. Further, simply applying the snaked proofreading transformation followed by the self-healing transformation, or vice versa, does not provide a solution (see the beginning of Section 4.4). In this paper we describe a new construction that has the same provable properties as snaked proofreading for the first type of error, but is also able to heal damaged areas where tiles have been removed from the assembly, even when errors in tile addition are allowed.

We assume the reader is familiar with the formal details of the Tile Assembly Model (see Chapter 2 for a long version, or Chapter 3 for a short summary). In the next section we review the model of the dynamics of self-assembly that allows us to speak more precisely about the rate of incorrect tile additions and to show that our construction is robust to such errors. Further, we'll specify more precisely the kind of damage we allow to our assemblies in studying the self-healing property. In the final section, we introduce our construction and prove that it is robust to both types of error. Our proof technique provides an alternative way of analyzing the error correction process in that all analysis pertains to individual blocks.

4.3 Modeling Errors

4.3.1 Erroneous Tile Additions During Growth

To be able to discuss whether or not a tile set is robust to erroneous tile additions, we need a model of the process of incorporation of erroneous tiles into the growing structure. In physical realizations of self-assembly, the growth process involves tiles dynamically attaching and detaching from the assembly. An error occurs if a tile that is held on with total strength less than 2 does not fall off quickly enough and becomes effectively locked in place when another tile attaches such that both tiles are now held on to the rest of the structure with strength at least 2. We term this event an *insufficient attachment*. Thus to determine the effective rate of insufficient attachments we need to study the dynamics of tile attachments and detachments.

Following [22, 24, 18] let us define the kinetic Tile Assembly Model (kTAM) as follows. The concentration of each tile type in solution is held constant throughout the self-assembly process, and the concentrations of all tile types are equal. We assume that for every tile association reaction there is a corresponding dissociation reaction. We further assume that the rate of addition (*forward rate* f) of any tile type at any position of the perimeter of the growing assembly is the same. Specifically, $f = k_f e^{-G_{mc}}$ where k_f is a constant that sets the time scale, and G_{mc} is the logarithm of the concentration of each tile type in solution. The rate that a tile falls off the growing assembly (*reverse rate* r_b) depends exponentially on the number of bonds that must be broken. Specifically, $r_b = k_f e^{-bG_{se}}$ where b is the total interaction strength with which the tile is attached to the assembly, and G_{se} is the unit bond free energy, which may depend, for example, on temperature*.

We assume the following concerning f and r_b . As in [22], we let $f \approx r_2$ since then the tile addition requirement imposed by the abstract Tile Assembly Model is satisfied with high probability, yet forward growth can still occur.[†] In Section 4.5.1 we discuss how close f and r_2 have to be for our proof to work out, but for the purposes of the rest of the paper we assume $f = r_2$. We assume f (and therefore r_2) can be arbitrarily chosen in our model by changing G_{mc} and G_{se} , for example by changing tile concentrations and temperature. (In practice, there are limits to how much these parameters can be changed.) However, k_f is assumed to be a physical constant not under our control.

Following [4, 18] we make use of a simplification of the kTAM that captures the essential behavior while being more tractable for rigorous proofs. Under the conditions where $f = r_2$, the self-assembly process is dominated by tiles being added with exactly 2 bonds and tiles falling off via exactly 2 bonds. The **locking** kTAM model assumes that these are the only possible single-tile events. That is, $r_b = 0$ for $b \geq 3$, and tiles

*This formulation ignores the initiation free energy of hybridization, which is non-negligible. See [22] for details of how this free energy can be treated, yielding a model that is formally identical, but with slightly altered physical meanings for G_{mc} and k_f .

[†]Assuming $f = r_2$, since $r_1 \gg f$, if a tile is added that bonds only with strength 1, it falls off very quickly as it should to obey the aTAM. Tiles attached with strength 2 stick much longer, allowing an opportunity for other tiles to attach to them. Once a tile is bonded with total strength 3, it is very unlikely to dissociate (unless surrounding tiles fall off first). Requiring $f > r_2$ ensures that on average, crystals grow, and for fixed r_2 , choosing $f \approx r_2$ minimizes the probability of an insufficient attachment. We will encounter additional reasons for choosing $f \approx r_2$ later.

never attach via a single weak (strength-1) bond. Additionally, insufficient attachments are modeled in the locking kTAM as atomic events, in which two tiles are added simultaneously at any position in which an insufficient attachment can occur. Specifically, any particular pair of tile types that can create an insufficient attachment in the kTAM is added at a rate $f_{err} = O(e^{-3G_{se}})$. (This is asymptotically the rate that insufficient attachments occur in the kTAM [4].) Thus the total rate of insufficient attachments at a particular location is Qf_{err} , where Q is the number of different ways (with different tile types) that an insufficient attachment can occur there. These insufficient attachments are the sole cause of errors during growth.

If a tile set is robust to insufficient attachments, then we can set G_{se} and G_{mc} such that the assembly grows quickly enough, yet the assembly will itself correct the errors caused by insufficient attachments.

4.3.2 Wholesale Removal of Tiles

Let us now consider how to model the event when (potentially large) portions of the completed pattern are physically ripped off the assembly despite being strongly bonded to it. We simply suppose that any number of tiles can be spontaneously removed from the assembly in a distinct event. However, we assume the L-shaped boundary tiles cannot get removed. If the assembled structure becomes disconnected after the event, we assume that the part of the assembly containing the L-shaped boundary remains.

The reason we suppose that the L-shaped boundary cannot get detached is that to make the boundary self-healing requires a different self-healing transformation than the one shown in Fig. 4.2 (see [23]), and we wish to keep our argument as simple as possible. It remains an open question whether the self-healing/proofreading construction presented in this paper can be extended to recover the boundary after damage, and whether the techniques used here can be extended to a wider class of tile sets that perform complex growth to create shapes and patterns (see Chapter 2). We expect an affirmative answer.

4.4 Self-Healing Proofreading Construction

First, let us return to the following issue raised in the Introduction: Why can't we simply apply the snaked proofreading transformation followed by the self-healing transformation, or vice versa, to produce a tile set robust to both insufficient attachments and wholesale removal of tiles? There are two difficulties. The first is of a technical nature: both transformations shown in Fig. 4.2 only are defined if precursor tiles have weak bonds on all four sides, yet they result in tile sets that also involve both strong and null bonds. Thus the two transformations can't be composed. Sufficiently general (though more complicated) self-healing transformations do exist [23], but although more generally applicable proofreading transformations have been proposed [24], there are as yet none with provably good performance. Even supposing this technicality can be overcome, there is no guarantee that the tile set resulting from composing both transformations will retain both robustness properties. One problem is that no matter in which order the transformations are applied, the blocks produced by the last transformation are sensitive to even one insufficient attachment after wholesale removal of tiles. Fig. 4.3 illustrates two incorrect structures that can form and become locked in (according to the locking kTAM). Therefore, we choose to combine the ideas from the snaked proofreading and self-healing constructions, and do not simply compose the transformations directly.

Our self-healing proofreading construction is illustrated in Fig. 4.4.

Suppose we are trying to assemble an $N \times N$ initial portion of the given pattern such that it assembles quickly and correctly with some fixed high probability (like 99%) from the starting L-shaped boundary or from any subassembly that may be formed by removing tiles from the target assembly. We have the following result*:

Theorem 4.4.1. *Fix any constant $\varepsilon > 0$, and consider any N . There is a $k = \Theta(\log N)$ such that using the self-healing proofreading construction with block size k and an L-shaped boundary N blocks long, with an appropriate choice of G_{mc} and G_{se} , the following holds in the locking kTAM model. Starting with any subassembly of A_N containing the L-shaped boundary, with probability at least $1 - \varepsilon$, the initial $N \times N$ block portion of the pattern A_N completes correctly in time $O(N \text{ poly}(\log N))$.*

*We use the standard asymptotic notation defined as follows: $f(x) = O(g(x))$ means that there is $c > 0$ such that $f(x) \leq c \cdot g(x)$ for large enough x . Similarly, $f(x) = \Omega(g(x))$ means that there is $c > 0$ such that $f(x) \geq c \cdot g(x)$ for large enough x . We write $f(x) = \Theta(g(x))$ if $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$.

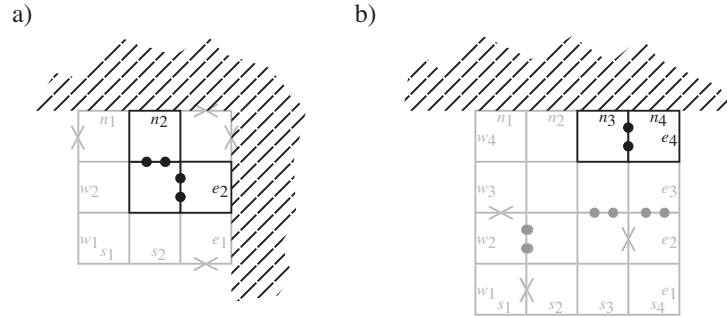


Figure 4.3: (a) The self-healing and (b) the snaked proofreading blocks are sensitive to a few insufficient attachments in the backward growth direction. Consider the case where in the original tile set, backward growth is not deterministic. The structures shown can form after a single insufficient attachment and may be incorrect since they involve backward growth. Every one of the tiles is attached with strength at least 3 and thus cannot dissociate in the locking kTAM. The striped areas show a part of the remaining (correct) assembly after wholesale removal of tiles. The grayed out tiles, which need not be present, show the entire block for reference. There are other structures that can form that cause similar problems.

As a special case, the subassembly we start out with may just be the L-shaped boundary. Then the assembly process we are talking about is the regular case considered by Chen and Goel which starts out from this boundary. However, we also consider the case where the assembly A_N was damaged by the removal of some tiles. Note that the assumption is that this damage is an isolated event rather than occurring continuously at some rate. If wholesale damage events occur less frequently than the time required to complete the $N \times N$ square, then a reasonable probability of success can be inferred. However, if damage occurs at an intermediate time during assembly — when many incorrect tiles are still present before being replaced by correct growth — then we need a stronger theorem that states that such initial conditions are also allowed. As this requires technical language defining just how much incorrect growth is allowable, we defer this discussion until the end of the proof of the stated theorem.

Proof. (of Theorem 4.4.1) First we need to make some terms precise. Then we will prove a number of lemmas about the assembly process, and finally with their help we will prove the theorem. In the following, when we say *block* we will mean the square $k \times k$ region which should become filled by a block in the ideal error-free assembly. We say a tile in an assembly is *incorrect* if it is not the same tile type as in the ideal error-free assembly we are trying to produce. Of the directions {north, east, south, west}, we will call the directions west and south the *input* directions, and east and north the *output* directions (because the growth direction is north-east and thus information must pass from the west and south toward north and east). We say that a block or a region becomes *clean* if all incorrect tiles detach (correct tiles may remain.)

Now refer to Fig. 4.4(b) where rectangles N , E , S , W are defined. We define the following in relation to a particular block in a growing assembly:

- State *DOOM*: The block enters this state when an input rectangle W or S touches an output rectangle N or E or if any of the rectangles touches the “spine” of the block (marked with wavy patterns in the figure). We will see that unless DOOM occurs, all of the rectangles are easy to clean. If DOOM occurs, this can no longer be guaranteed and indeed the block can lock in with incorrect tiles.
- Event *IA*: This event occurs whenever an insufficient attachment happens in the block or its input blocks.
- State *CLEAN*: This state occurs when the block becomes clean, together with the abutting output rectangles of its input blocks. We will demonstrate that after a CLEAN, many IA events are required to enter the DOOM state.
- State *COMPLETE*: The block enters this state when it and its input blocks complete correctly. We will see that once a block enters this state the correct structure is locked in and we can move on to other

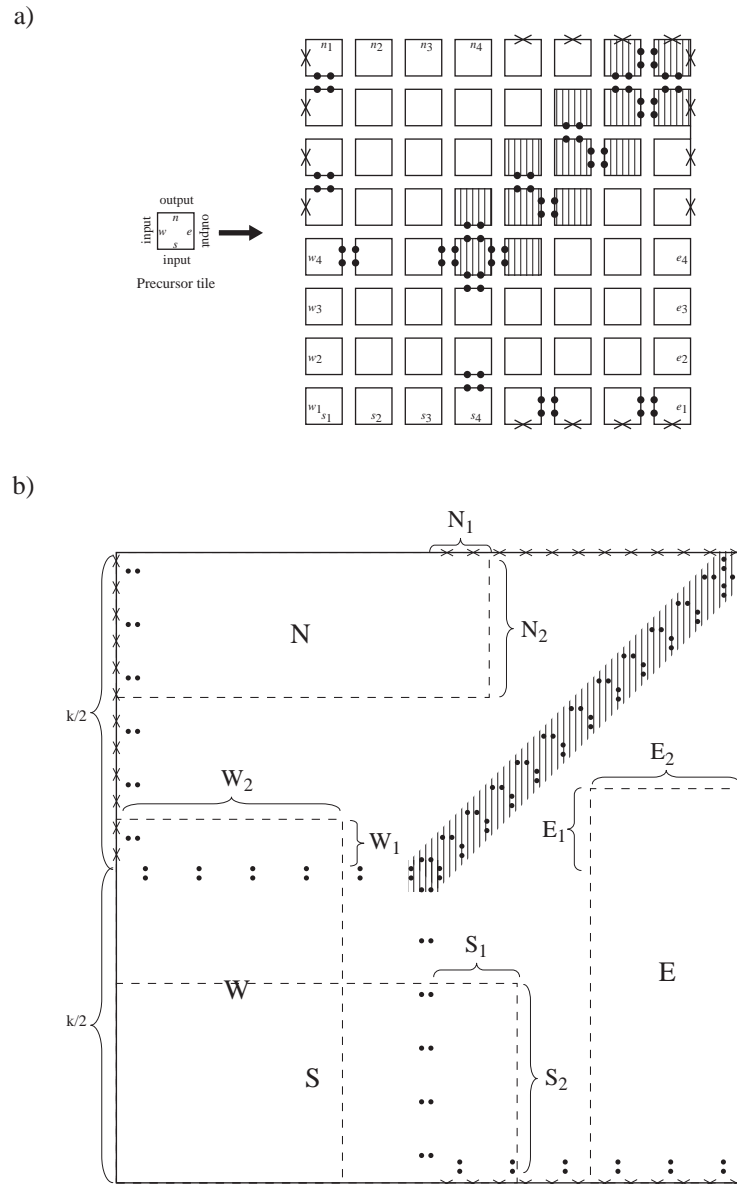


Figure 4.4: (a) The self-healing proofreading transformations for block size $k = 8$. (b) Schematic of the self-healing proofreading block for k divisible by 4. Tiles in the $k \times k$ block are not explicitly drawn; just the pattern of null bonds (Xs) and strong bonds (double-dots) are indicated, and all other bonds are weak. For discussing the growth process in the presence of errors, the state of the assembly is characterized by N_i , E_i , S_i , and W_i , which are the smallest non-negative integers such that the following statements hold: Rectangle N (E) contains all incorrect tiles connected to the north (east) side of the block. (We say that a tile is connected to a side of a block if there is a path of connected tiles (abutting with matching bond types) within the block from the given tile to the given side.) Rectangle S (W) contains all incorrect tiles connected to an input side of the block that are incorrect with respect to the west (south) side of the precursor tile. (An incorrect tile must be incorrect with respect to at least one of the input sides of the precursor tile.) Wavy patterns indicate tiles forming the block “spine”.

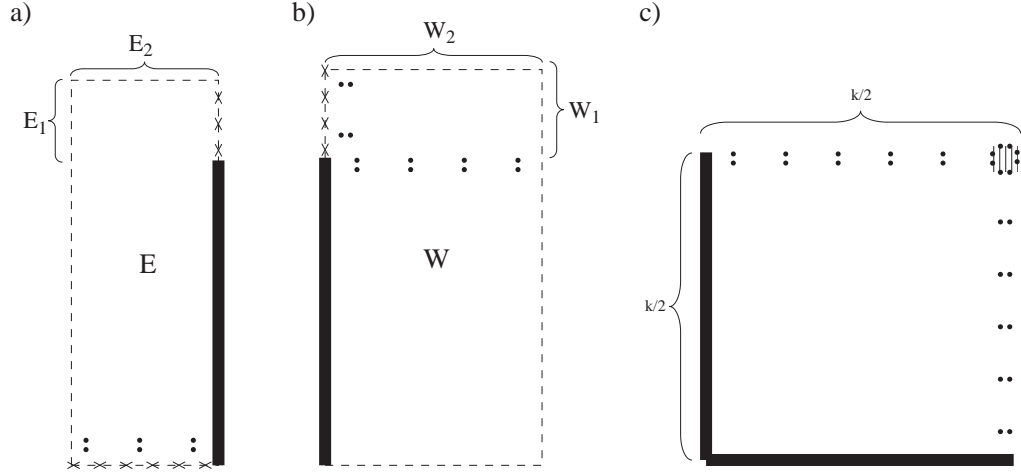


Figure 4.5: (a,b) Illustration for the proof of Lemma 4.4.3. (c) Illustration for the proof of lemma 4.4.4. In (a)–(b) the thick black lines indicate where the incorrect tiles defining the rectangles shown may be bonded to the rest of the assembly (and conversely the dashed lines indicate where they may not be bonded to the rest of the assembly). In (c) the thick black lines indicate where the correct tiles are bonded to the rest of the assembly.

blocks in our argument.

Lemma 4.4.1. *If a block is COMPLETE then no tile from the block can detach.*

Proof. By inspection: except for the most south-west tile, every tile in a completed block is attached by strength at least 3. Assuming both input blocks are completed, the most south-west tile is also attached by strength at least 3. \square

Lemma 4.4.2. *If a block is CLEAN then (a) at least one IA is needed to get an incorrect tile in the block, (b) at least $k/4$ IA events are needed to enter DOOM, assuming no DOOM occurs in its input blocks first.*

Proof. Part (a) follows immediately. Unless a DOOM occurs in our block or its input blocks, each insufficient attachment inside our block increases one of $N_2, E_2, S_1, S_2, W_1, W_2$ by at most 2 or one of N_1, E_1 by at most 1. An insufficient attachment in the west or south input block can increase S_2 or W_2 respectively by 2 (if the incorrect tiles extend into the input blocks). Thus, the number of IAs must be at least $N_1 + E_1 + (N_2 + E_2 + S_1 + S_2 + W_1 + W_2)/2$. At least one of the following inequalities must hold for DOOM to occur: $N_1 + N_2 \geq k/2 - 1, E_1 + E_2 \geq k/2 - 1, W_2 \geq k/2 - 1, S_2 \geq k/2 - 1, W_1 + N_2 \geq k/2$, or $S_1 + E_2 \geq k/2$. Part (b) easily follows. \square

Lemma 4.4.3. *The expected time for a block to enter CLEAN is $O(k^3/f)$, assuming (1) no IA occurs, and (2) no DOOM occurs in this block or its input blocks.*

Proof. Let us first show that the output rectangles of the input blocks become clean in expected time $O(k^3/f)$. Let's consider rectangle E since N can be treated identically. Since no DOOM occurs in this block, we can safely assume that these incorrect tiles are surrounded by either correct tiles or empty space on the north and west, and thus cannot bind to them. Then the largest incorrect structure is illustrated in Fig. 4.5(a). Recall that we are assuming that the forward rate f is equal to the reverse rate r_2 (see Section 4.3.1). Thus the leftmost two rows can fall off via a 1-D random walk with expected time $O(k^2/f)$ (see Section 4.5.1). Once the two rows fall off, they cannot attach again except via another insufficient attachment. Since there are $O(k)$ pairs of rows, the total expected time for rectangle N to fall off is $O(k^3/f)$.

Once the output rectangles of the input blocks become clean, only correct (or empty) tiles abut our block on the west and south sides. Let's consider the tiles defining rectangle W (rectangle S can be treated similarly). Since these tiles are incorrect with respect to the south side of the block, they cannot be attached

to anything on the south side of the rectangle. Further they cannot be attached to anything inside the block along the east or north sides of the rectangle since then those tiles would be part of the rectangle. Since we are assuming that no DOOM occurs, the rectangle cannot extend to and bind the north output border of the block either. Further, the rectangle cannot reach the column of strength-2 bonds on its right because otherwise a block error would occur (a spine tile would be covered by the rectangle). Thus the rectangle W is as illustrated in Fig. 4.5(b). The $W_2 \times W_1$ top rectangle can fall off via 1-D random walks as before. After that, again by the same argument, the rest of rectangle W can fall off in time $O(k^3/f)$. \square

Lemma 4.4.4. *The expected time for a block whose input blocks are COMPLETE to enter COMPLETE itself is $O(k^3/f)$, assuming (1) no IA occurs, and (2) no DOOM occurs.*

Proof. First the block enters CLEAN in expected time $O(k^3/f)$ using Lemma 4.4.3 (note that no DOOM can occur in the input blocks because they are completed). By Lemma 4.4.2(a) the block remains free of incorrect tiles. Then let us consider how long it takes to complete the cleaned block whose input blocks are complete, assuming no insufficient attachments occur. Consider the south-west quarter of the block shown in Fig. 4.5(c). Once each row or column completes it is held by strength at least 3 and thus cannot dissociate. Each row or column completes via a 1-D random walk with expected time $O(k^2/f)$. Since there are $O(k)$ row/columns, the total expected time to complete the square shown is $O(k^3/f)$. The remaining areas of the block can be completed using a similar argument in time $O(k^3/f)$ as well, after the spine of the block (wavy pattern in Fig. 4.4) completes in time $O(k/f)$. \square

Now using these lemmas we can finish the proof of Theorem 4.4.1. The argument will proceed as follows. First, we set $e^{-G_{se}}$ low enough as a function of the block size k so that insufficient attachments are sufficiently rare that a block has a high probability of entering CLEAN or COMPLETE before an IA occurs. This will ensure that, assuming no DOOM occurs anywhere, the assembly completes in time $O(\text{poly}(k)N/\varepsilon)$. Then we will set k large enough as a function of N and ε to ensure that no block enters the DOOM state during the entire assembly process. We will see that k need not be more than logarithmic in N/ε .

Recall that as long as a particular location remains susceptible, we model insufficient attachments at that location as a Poisson process with rate $O(Qe^{-3G_{se}})$ where Q is the number of different tile types that can be added via an insufficient attachment there. Q can be bounded by the total number of different (made up of different tiles) blocks, and since this is the number of tile types in the original tile system, it does not change with k and can be absorbed into the constant. Thus for any block, the distribution of the time until an IA occurs can be upper bounded by an exponential random variable with expected time $t_{ia} = \Omega(1/(k^2e^{-3G_{se}}))$ since there are no more than $3k^2$ locations where an insufficient attachment can occur (in the block and its input blocks). Let $t_c = O(k^3/f)$ be the worst case expected time for Lemmas 4.4.3 and 4.4.4 (over all possible assemblies, blocks and configurations of tiles in blocks). We will want that $t_c \leq (1/2)t_{ia}$. Recalling that $f = r_2 = O(e^{-2G_{se}})$ (Section 4.3.1), we can guarantee that this inequality is satisfied if we set $e^{-G_{se}}$ low enough: $e^{-G_{se}} = O(1/k^5)$. However, setting $e^{-G_{se}}$ too low slows down the rate of assembly more than necessary, and thus for the following we assume that $e^{-G_{se}} = \Theta(1/k^5)$. Then, $t_c = O(k^{13})$.

We wanted to have $t_c \leq (1/2)t_{ia}$ in order to show that the $N \times N$ blocks are likely, with probability at least $1 - \varepsilon/2$, to assemble correctly in time $O(\text{poly}(k)N/\varepsilon)$ assuming no DOOM occurs anywhere. This can be argued as follows. Consider any block whose input blocks are COMPLETE. Lemma 4.4.4 lets us bound the time until the block itself COMPLETEs assuming no IAs or DOOM occur. But what if IAs can occur? The probability that COMPLETE occurs within time $2t_c$ given that an IA doesn't happen in this time is at least $1/2$ by Lemma 4.4.4 and the Markov inequality.* The probability that an IA doesn't happen in this time is at least $1/e$ since $2t_c \leq t_{ia}$. Thus the probability that COMPLETE occurs within time $2t_c$ is at least $(1/2)(1/e) = 1/(2e)$. If it doesn't (i.e., an IA occurs or it simply doesn't finish), the probability that a COMPLETE will occur in the next $2t_c$ interval is again at least $1/(2e)$. Thus the expected time until COMPLETE occurs is at most $2e2t_c = 4et_c$. Recall that once a block completes, it can't fall apart (Lemma 4.4.1). Thus, the current situation is equivalent to irreversible, error-free self-assembly of tiles, where each tile represents a block in our system. In irreversible assembly, the time to assemble an $N \times N$ tile square from an L boundary is on the order of N times the expected time for a single tile to attach [1]. Thus,

*The Markov inequality states that for any non-negative random variable X , $Pr[X \geq a] \leq E[X]/a$ where $E[X]$ is the expected value of X .

the expected total time to complete the $N \times N$ block assembly is $t_{tot} = O(N \cdot 4et_c) = O(Nt_c)$ assuming no DOOM occurs anywhere. Therefore, the probability that it takes longer than $t_{max} = t_{tot}(2/\varepsilon) = O(Nk^{13}/\varepsilon)$ to complete the assembly or to get a DOOM somewhere is at most $\varepsilon/2$, again by the Markov inequality.

Next we bound the probability that DOOM occurred anywhere in the assembly in time t_{max} . We'll show that by an appropriate choice of $k = \Theta(\log(N/\varepsilon))$, the probability of this happening is no more than $\varepsilon/2$. Again focus on a particular block; but this time the two input blocks may not be completed. Let us make the worst case assumption that the block remains uncompleted for the duration of assembly t_{max} and thus susceptible to DOOM. We want such a block to be without DOOM for the entire time. Recall that the expected time until an IA is bounded by $t_{ia} = \Omega(1/(k^2e^{-3G_{se}}))$. Thus even with the worst case assumption that the block is never completed, the expected number of IAs for this block in time t_{max} is at most $q = O(t_{max}k^2e^{-3G_{se}})$. Recalling that $e^{-G_{se}} = O(1/k^5)$, we have $q = O(N/\varepsilon)$. The probability that there will be more than $qN^2(4/\varepsilon)$ is at most $\varepsilon/(4N^2)$ by the Markov inequality. After each IA occurs, with probability at least $1/(2e)$ there will be a CLEAN but no IA within time $2t_c$ (using the same argument as we followed in the previous paragraph for COMPLETE). Thus with probability at least $1/(2e)$, a CLEAN will occur between two IAs. So the probability that among no more than $qN^2(4/\varepsilon)$ IAs, a run of $k/4$ occur in a row without intervening CLEANs can be upper-bounded by $p_{run} = qN^2(4/\varepsilon)(1 - 1/(2e))^{k/4}$. Since for DOOM to occur, $k/4$ IAs must occur without intervening CLEANs (Lemma 4.4.2(b)), the probability of DOOM in this block during the entire assembly time is upper bounded by p_{run} if no more than $qN^2(4/\varepsilon)$ IAs occur. If we can somehow guarantee that $p_{run} \leq \varepsilon/(4N^2)$, then the probability that DOOM occurs in this block during the entire assembly time t_{max} is at most $\varepsilon/(4N^2) + \varepsilon/(4N^2) = \varepsilon/(2N^2)$. Since there are N^2 blocks, the probability that DOOM will occur even in one of them during the entire assembly time t_{max} is at most $\varepsilon/2$.

Now all that is left is to guarantee that $p_{run} = qN^2(4/\varepsilon)(1 - 1/(2e))^{k/4} \leq \varepsilon/(4N^2)$. Solving for k , we get: $k \geq O(1) \log(16N^4q/\varepsilon^2) = O(\log(N/\varepsilon))$.

Recall that the total assembly time $t_{max} = O((N/\varepsilon)k^{13}(Q(k))^2)$. Using $k = O(\log(N/\varepsilon))$, we get that $t_{max} = O(N \cdot \text{poly}(\log N))$, for any fixed ε . \square

Note that Theorem 4.4.1 can be strengthened to allow incorrect tiles in the starting subassembly, as long as there is no DOOM in any of the blocks. Thus we can cover the case in which the assembly process does not entirely complete before the wholesale removal of tiles occurs. However, if this removal occurs periodically and large enough portions of the assembly are taken out each time, it may be that the assembly is never given a chance to complete. In any case DOOM will eventually prevail.

4.5 Extensions

4.5.1 Random Walks in the $r_2 \neq f$ Regime

Our proofs require that $f = r_2$ exactly. This can't be achieved exactly in practice, which begs the question, is it a necessary requirement for our construction and proof, and can it be relaxed somewhat? We believe it can be only slightly relaxed, due to the competing pressures of needing large patches of incorrect growth to be quickly removed, and at the same time needing correct growth to proceed quickly.

In the proofs of Lemmas 4.4.3 and 4.4.4, we model the completion and dissociation of a chain of sequentially added/removed tiles as a continuous time 1-D random walk, where the rate with which a tile is added at the end is f and the rate as which the last tile is removed is r_2 . Specifically, we rely on the fact that the expected time for the entire chain to complete (allowing fast forward growth) and the expected time for the chain to fall off (allowing errors to be quickly undone), are both fast (polynomial in the block size k and therefore logarithmic in the size of the target assembly N).

In order to compute the expected time until the entire chain is completed (or equivalently falls off) we can use the following argument. In the discrete time 1-D random walk of length $a = O(k)$, the expected number of steps to reach a predetermined end (with the other end being a reflective barrier) is $O(a^2)$ if the forward and backward transition probabilities are equal [7]. In our case, if $r_2 = f$, the two transition probabilities are indeed equal. Further, since the expected time to take one step is $1/(r_2 + f) = 1/(2f)$, the expected time to

reach the predetermined end is $O(a^2/f) = O(k^2/f)$.*

However, what happens if the forward rate f does not equal the reverse rate r_2 ? In the discrete time biased 1-D random walk of length a (with a reflecting barrier on the favored end), the expected number of steps to complete in the unfavored direction is $S(\gamma, a) = \frac{1}{2\gamma^2}(1 + \gamma) \left[\left(\frac{1+\gamma}{1-\gamma} \right)^a - 1 \right] - a/\gamma$ where γ is the difference between the transition probability in the favored and the unfavored directions.[†] This expected time is monotonic increasing in γ and exponentially increasing in a . So if γ is not decreased as we attempt to build larger and larger portions of patterns requiring larger block sizes, then the average number of steps in this random walk grows exponentially with $a = O(k)$, which would not allow us to obtain Theorem 4.4.1.

Thus, as the block size increases, we need r_2 and f to be closer to each other. As a function of k (and thus ultimately of N) how fast does the difference need to decrease in order for Theorem 4.4.1 to still hold? Let us assume that G_{se} and thus r_2 is set as required by our proof, but we didn't get G_{mc} quite right such that the actual forward rate f is slightly smaller than r_2 . This would normally mean that crystals would be thermodynamically driven to shrink, but since some tile additions form multiple bonds, locking the tile in, assembly is still ratcheted forward. The rate of insufficient attachments can only be smaller and thus still $f_{err} = O(e^{-3G_{se}})$. Thus as long as we can still prove Lemmas 4.4.3 and 4.4.4 we would be done. Observe that $\gamma = \frac{r_2-f}{r_2+f}$ is the difference between the transition probabilities in the favored and unfavored directions in the corresponding discrete time 1-D random walk. Assuming that γ decreases at least as fast as $1/a$, the expected number of steps of the discrete time Markov process to complete in the unfavored direction is no more than $S(\frac{1}{a}, a) = \frac{1}{2}(a^2 + a) \left[\left(\frac{a+1}{a-1} \right)^a - 1 \right] - a^2 = O(a^2)$, since $\lim_{a \rightarrow \infty} \left(\frac{a+1}{a-1} \right)^a = e^2$. This implies that the expected time for the continuous-time Markov process to complete in the unfavored direction is still $O(k^2/f)$, as required for Lemmas 4.4.3 and 4.4.4, as long as γ decreases at least as fast as a function in $O(1/k)$.

A thermodynamic argument based on a more realistic kTAM model may require γ to decrease slightly faster, however. In the full kTAM [22], in which every reaction has a reverse reaction and an equilibrium-satisfying detailed balance can be defined, growth of blocks is biased forward if the free energy of adding an entire $k \times k$ block is favorable. This free energy may be calculated as $\Delta n G_{mc} - \Delta b G_{se}$, where Δn is the number of tiles added, and Δb is the total strength of all new bonds. In our construction, adding a block entails $\Delta n = k^2$ and $\Delta b = 2k^2 + 2$. Thus, favorable growth requires that $\frac{G_{mc}}{G_{se}} < 2 + \frac{2}{k^2}$. Now, since neatly $\gamma = \frac{r_2-f}{r_2+f} = \tanh\left(\frac{G_{mc}-2G_{se}}{2}\right)$, the favorable growth condition requires that $\gamma < \tanh(G_{se}/k^2)$. Since the proof of Theorem 4.4.1 required that $e^{-G_{se}} = \Theta(1/k^5)$, $G_{se} = \Theta(\log k)$ and thus the favorable growth condition reduces to $\gamma = O(\tanh(\log k/k^2))$. This is slightly more strict than $\gamma = O(1/k)$ derived in the locking kTAM above.

4.5.2 Preventing Spurious Nucleation

The blocks produced by our construction have large regions in which tiles are connected to each other via strong (strength 2) bonds (i.e., the ‘‘spine’’ of the block, wavy pattern in Figure 4.4). When the constituent tiles are placed in solution, there is a danger that they will spontaneously nucleate and growth will proceed disconnected from the seed L-shaped boundary. Further, growth may proceed by the aggregation of the separately-nucleated fragments. In other words, our model assumptions that only the assembly containing the L-shaped boundary will grow, and that it will grow by single tile additions, may be violated in practice for such tile sets. Can we avoid large regions of strongly bonded tiles in our construction? We believe ‘‘zig-zag’’ boundaries [16] can be adopted to replace the spine, although details remain to be worked out. Rather than a fixed-width spine, this spine would need to be thicker to be more and more robust to spurious nucleation.

* At the reflecting barrier the expected time to take a step is twice as large since only the forward direction is possible. However, this does not affect the asymptotic results.

[†] See [7] for the general form of the expected duration of 1-D discrete time random walks, from which the above expression is derived.

Acknowledgments:

We thank Ho-Lin Chen and Ashish Goel for insightful conversations and suggestions. This work was supported by NSF Grant No. 0523761.

Bibliography

- [1] Leonard M. Adleman, Qi Cheng, Ashish Goel, and Ming-Deh A. Huang. Running time and program size for self-assembled squares. In *ACM Symposium on Theory of Computing (STOC)*, pages 740–748, 2001.
- [2] Gagan Aggarwal, Qi Cheng, Michael H. Goldwasser, Ming-Yang Kao, Pablo Moisset de Espanés, and Robert T. Schweller. Complexities for generalized models of self-assembly. *SIAM Journal on Computing*, 34:1493–1515, 2005.
- [3] Robert D. Barish, Paul W. K. Rothmund, and Erik Winfree. Two computational primitives for algorithmic self-assembly: Copying and counting. *NanoLetters*, 5:2586–2592, 2005.
- [4] Ho-Lin Chen and Ashish Goel. Error free self-assembly using error prone tiles. In Ferretti et al. [8], pages 62–75.
- [5] Junghuei Chen and John Reif, editors. *DNA Computing 9*, volume LNCS 2943. Springer-Verlag, 2004.
- [6] Matthew Cook, Paul Wilhelm Karl Rothmund, and Erik Winfree. Self-assembled circuit patterns. In Chen and Reif [5], pages 91–107.
- [7] W. Feller. *An introduction to probability theory and its applications. Vol. 1*. Wiley, 1968.
- [8] Claudio Ferretti, Giancarlo Mauri, and Claudio Zandron, editors. *DNA Computing 10*, volume LNCS 3384. Springer-Verlag, 2005.
- [9] Paul W. K. Rothmund, Nick Papakakis, and Erik Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology*, 2:e424, 2004.
- [10] Paul W. K. Rothmund and Erik Winfree. The program-size complexity of self-assembled squares. In *ACM Symposium on Theory of Computing (STOC)*, pages 459–468, 2000.
- [11] Thomas H. LaBean, Hao Yan, Jens Kopatsch, Furong Liu, Erik Winfree, John H. Reif, and Nadrian C. Seeman. Construction, analysis, ligation, and self-assembly of DNA triple crossover complexes. *Journal of the Americal Chemical Society*, 122:1848–1860, 2000.
- [12] Michail G. Lagoudakis and Thomas H. LaBean. 2-D DNA self-assembly for satisfiability. In Erik Winfree and David K. Gifford, editors, *DNA Based Computers V*, volume 54 of *DIMACS*, pages 141–154. American Mathematical Society, 2000.
- [13] Chengde Mao, Thomas H. LaBean, John H. Reif, and Nadrian C. Seeman. Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. *Nature*, 407:493–496, 2000.
- [14] Chengde Mao, Weiqiong Sun, and Nadrian C. Seeman. Designed two-dimensional DNA holliday junction arrays visualized by atomic force microscopy. *Journal of the Americal Chemical Society*, 121:5437–5443, 1999.
- [15] John Reif. Local parallel biomolecular computing. In Harvey Rubin and David Harlan Wood, editors, *DNA Based Computers III*, volume 48 of *DIMACS*, pages 217–254. American Mathematical Society, 1999.
- [16] Rebecca Schulman and Erik Winfree. Programmable control of nucleation for algorithmic self-assembly. In Ferretti et al. [8], pages 319–328. Extended abstract in *DNA Computing 10*; preprint of the full paper is cond-mat/0607317 on arXiv.org.

- [17] Rebecca Schulman and Erik Winfree. Self-replication and evolution of DNA crystals. In Mathieu S. Capcarrere, Alex A. Freitas, Peter J. Bentley, Colin G. Johnson, and Jon Timmis, editors, *Advances in Artificial Life: 8th European Conference (ECAL)*, volume LNCS 3630, pages 734–743. Springer-Verlag, 2005.
- [18] David Soloveichik and Erik Winfree. Complexity of compact proofreading for self-assembled patterns. In *DNA Computing 11*. Springer-Verlag, 2005.
- [19] David Soloveichik and Erik Winfree. Complexity of self-assembled shapes, 2005. Extended abstract in DNA Computing 10; preprint of the full paper is cs.CC/0412096 on arXiv.org.
- [20] Erik Winfree. On the computational power of DNA annealing and ligation. In Richard J. Lipton and Eric B. Baum, editors, *DNA Based Computers*, volume 27 of *DIMACS*, pages 199–221. American Mathematical Society, 1996.
- [21] Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, 1998.
- [22] Erik Winfree. Simulations of computing by self-assembly. Technical report, Caltech, 1998.
- [23] Erik Winfree. Self-healing tile sets. In Junghuei Chen, Natasha Jonoska, and Grzegorz Rozenberg, editors, *Nanotechnology: Science and Computation*, pages 55–78. Springer-Verlag, 2006.
- [24] Erik Winfree and Renat Bekbolatov. Proofreading tile sets: Error-correction for algorithmic self-assembly. In Chen and Reif [5], pages 126–144.
- [25] Erik Winfree, Furong Liu, Lisa A. Wenzler, and Nadrian C. Seeman. Design and self-assembly of two dimensional DNA crystals. *Nature*, 394:539–544, 1998.
- [26] Erik Winfree, Xiaoping Yang, and Nadrian C. Seeman. Universal computation via self-assembly of DNA: Some theory and experiments. In Laura F. Landweber and Eric B. Baum, editors, *DNA Based Computers II*, volume 44 of *DIMACS*, pages 191–213. American Mathematical Society, 1998.

Chapter 5

The Computational Power of Benenson Automata

This chapter was published as: David Soloveichik and Erik Winfree, “The Computational Power of Benenson Automata,” *Theoretical Computer Science*, 244(2–3):279-297, 2005.

5.1 Abstract

The development of autonomous molecular computers capable of making independent decisions in vivo regarding local drug administration may revolutionize medical science. Recently Benenson et al. [4] have envisioned one form such a “smart drug” may take by implementing an in vitro scheme, in which a long DNA state molecule is cut repeatedly by a restriction enzyme in a manner dependent upon the presence of particular short DNA “rule molecules.” To analyze the potential of their scheme in terms of the kinds of computations it can perform, we study an abstraction assuming that a certain class of restriction enzymes is available and reactions occur without error. We also discuss how our molecular algorithms could perform with known restriction enzymes. By exhibiting a way to simulate arbitrary circuits, we show that these “Benenson automata” are capable of computing arbitrary Boolean functions. Further, we show that they are able to compute efficiently exactly those functions computable by log-depth circuits. Computationally, we formalize a new variant of limited width branching programs with a molecular implementation.

5.2 Introduction

The goal of creating a molecular “smart drug” capable of making independent decisions in vivo regarding local drug administration has excited many researchers [10]. Recently, Benenson et al. [4] (based on [5, 3]) have envisioned what such an automaton may look like, and reported a partial implementation of the design in vitro. They made a system consisting of an enzyme and a set of DNA molecules which tests whether particular RNA molecules are present in high concentration and other particular RNA molecules are present in low concentrations, and releases an output DNA molecule in high concentration only if the condition is met. The actual computation process consists of the enzyme cutting a special DNA molecule in a manner ultimately determined by the concentrations of input mRNA molecules present in solution. The authors suggest that such a design, or a similar one, can be used to detect concentrations of specific mRNA transcripts that are indicative of cancer or other diseases, and that the output can take the form of a “therapeutic” ssDNA.

The key computational element in the scheme is an enzyme that cuts DNA in a controlled manner. Nature provides many biologically realizable methods of cutting DNA that can be adapted for computing. For instance, bacteria have evolved methods to cut the DNA of invading viruses (phages) with numerous enzymes called restriction enzymes. Most restriction enzymes cut double-stranded DNA exclusively at sites where a specific sequence, called the recognition site, is found. Some restriction enzymes leave a so-called “sticky end overhang” which is a region of single-stranded DNA at the end of a double-stranded DNA molecule.

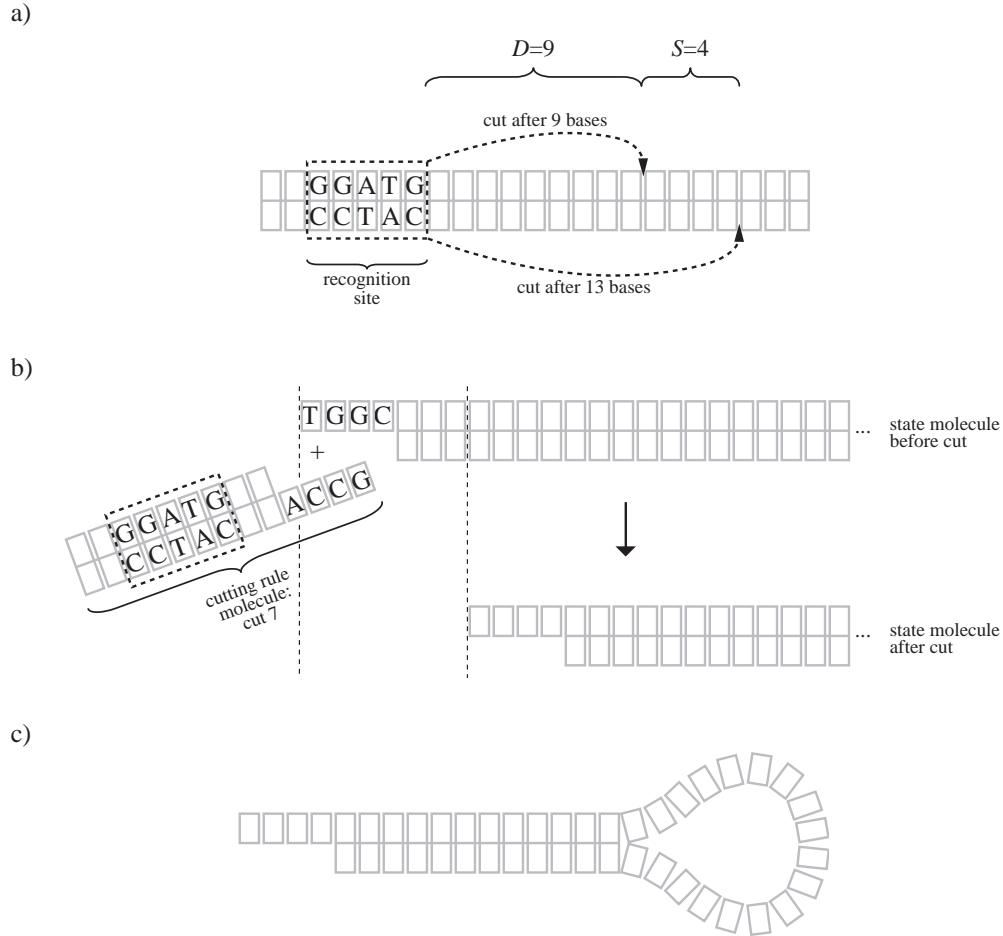


Figure 5.1: (a) *FokI* recognition and cut sites on a generic DNA substrate. The parameters D and S will be used to characterize restriction enzymes in this paper. D is called the cutting range and S the sticky end size. (b) Example of a cutting rule application. (c) Illustration of the output loop. Cutting far enough opens the loop. (In (a),(b),(c) the top strand is 5'→3'.)

Sticky ends are important because if there is another DNA molecule with a complementary sticky end, the two molecules can bind to each other forming a longer double-stranded DNA strand.

Benenson et al. use type IIS restriction enzymes, which cut double-stranded DNA at a specific distance away from their recognition sites in a particular direction [11]. These enzymes were first considered in molecular computation by Rothemund [6] in a non-autonomous simulation of a Turing machine. For an example of a type IIS restriction enzyme, consider *FokI* which is known to cut in the manner shown in Fig. 5.1(a). Note that after *FokI* cuts, the DNA molecule is left with a sticky end overhang of 4 bases. The automaton of Benenson et al. is based on a series of restriction enzyme cuts of a long *state molecule*. Each cut is initiated by the binding of a *cutting rule molecule* to the state molecule via matching sticky ends (Fig. 5.1(b)). Cutting rule molecules have an embedded restriction enzyme recognition site at a certain distance from their sticky end. The number of base pairs between the restriction enzyme recognition site and the sticky end on the cut rule molecule determines the number of bases that are cut away from the state molecule after the rule molecule attaches. Since the sequence of the sticky end on the state molecule determines which rule molecule attaches, it determines how many bases are cut off the state molecule in the presence of some set of rule molecules. Fig. 5.1(b) illustrates how TGGC can encode the “cut 7 bases” operation when the appropriate cutting rule molecule is present. After each cut, a new sticky end is revealed which encodes the location of the next cut, and the process can continue.

Benenson et al. [4] describe how any set of RNA or DNA molecules can act as input to their automaton.

In particular, each input species converts some rule molecules that are initially inactive into active form, and inactivates others that are initially active. The net effect of multiple pre-processing steps is that the presence of input molecules in either high or low concentration determines which rule molecules will be available. Note that input is provided all at once, at the beginning of the computation; the activated rule molecules are used by the automaton as needed during the course of the computation.

A single-stranded loop is attached to the end of the state DNA molecule (Fig. 5.1(c)). The loop is held closed by the remaining double-stranded part of the state molecule — the so-called *stem*. If the state molecule is cut close enough to the loop, the loop is opened and released. Assuming the loop has a chemical function only when open (e.g., it is translated to create a protein or effectively acts as antisense DNA), this results in the production of the “therapeutic” molecule in an input-dependent manner. If the system worked without error, and supposing that the input RNA molecules are either present in high concentration or not at all, the output DNA molecule should be released if and only if a set of RNAs is present that results in a set of rule molecules that cut the state DNA molecule sufficiently far. To accommodate the possibility of error, which we ignore here, Benenson et al. implement two possible outputs that compete between each other, with the one produced in largest quantities “winning.”

We are interested in the class of computations that can be implemented using the approach developed by Benenson et al. [4]. One possibility of performing complex computations using this scheme is to use the output DNA molecule of one Benenson automaton as an input for another, allowing feed-forward circuits to be implemented. However, we would like to study the computational power of a system with a single state molecule. Showing how to compute complex functions with a single Benenson automaton examines the computational power of the basic unit of computation, and makes it clear how one can compute even more complex functions with many state molecules.

In the first part of this paper, we formalize the computational process implemented by Benenson et al. using a system with a single state molecule. As part of our abstraction, we are going to ignore concentration dependence and other analog operations such as those involving probabilistic competition between various reactions, and will focus on a binary model in which a reaction is either possible or not.* We treat the state molecule and the set of possible cutting rule molecules as a program specifying what computation is to be performed, while the input determines which rule molecules are active. Each rule molecule depends upon a specific input RNA species which either activates or deactivates it, or it may be always active. We’ll say that a Benenson automaton outputs 1 if at some point at least a total of p bases has been cut off, where p represents the point in the state string cutting beyond which opens the loop. Otherwise, we say it outputs a 0. Our constructions will cut the state molecule to leave no stem on a 1 output, and some length of stem otherwise.†

Like circuits, Benenson automata are best studied as a non-uniform computing model. But while the computational power of circuits is well characterized, the computational power of Benenson automata has not been studied. For example, while it was shown [4] how a single Benenson automaton can compute a conjunction of inputs (and negated inputs), it was not clear how a single Benenson automaton can compute a disjunction of conjunctions. While [5] and [3] show how finite automata can be simulated by a similar scheme‡, a different input method is used. Here, we show that a Benenson automaton can simulate an arbitrary circuit, implying that it is capable of doing arbitrary non-uniform computation.

Lastly we study the cost of implementing more complex computations (e.g., more complex diagnostic tests) using Benenson automata. While increasing the length of the state molecule is relatively easy and incurs approximately linear cost, increasing the size of the sticky ends or the range at which the restriction enzyme cuts requires discovering or creating new enzymes. Enzymes with very large cutting ranges that leave large sticky ends may not exist in nature, and while some success has been achieved in creating new restriction enzymes [8, 7], engineering new restriction enzymes suitable for Benenson automata will require

*We’ll consider non-deterministic computation in which more than one cutting rule molecule can attach and cut. However, unlike [1] we will not assign probabilities to the various reactions and the output.

†Even with a stem remaining, the loop may still open at a certain rate (the “stem” must be long enough to keep the loop locked closed — see [4]). Nevertheless, our constructions can be modified to assure a longer remaining stem on a 0 output at the cost of using a few additional unique sticky ends (see Discussion).

‡In contrast to [4], [5, 3, 1] treat the state molecule as an input string for a uniform computation, while the set of rule molecules is always the same and specifies the finite state machine computation to be performed. It is interesting to note the difference in the computational power of these two approaches. To implement a FSM with K symbols and N states, a type IIS restriction enzyme with cutting range N and sticky end size $O(\log KN)$ is sufficient and probably necessary.

further technological advances.

Let us consider a family of Boolean functions $\{f_n\}$, where $n = 1, 2, \dots$ and $f_n : \{0, 1\}^n \rightarrow \{0, 1\}$. We show that any $\{f_n\}$ can be computed by a family of Benenson automata such that the size of the sticky ends grows only logarithmically with n and the range of enzyme cutting stays constant. (This is analogous to noting that any $\{f_n\}$ can be computed by a family of circuits using constant fan-in/fan-out, but it is non-trivial to prove.) If we restrict the length of the state molecule to be $\text{poly}(n)$, then the families of functions computable by these Benenson automata are exactly those computable by $O(\log(n))$ depth circuits. These results are asymptotically optimal, since sticky end lengths must grow as $\log n$ in order to read all the input bits. We'll also show that allowing the sticky end size to grow faster than $O(\log n)$ does not increase computational power, and that allowing logarithmic cutting range cannot increase it significantly. Finally, we'll define non-deterministic computation and prove that function families cannot be computed more efficiently using non-deterministic Benenson automata than deterministic ones.

Independent of the relevance of our formalization to biological computation, Benenson automata capture a model of string cutting with input-dependent cutting rules, and may be of interest as such.

5.3 Formalization of Benenson Automata

We consider Benenson automata over a fixed alphabet Σ . For biological plausibility, one may want to consider $\Sigma = \{A, T, C, G\}$. However, our constructions assume only that $|\Sigma| \geq 3$. If so desired, all our results can be adapted to a binary alphabet by utilizing two bits to represent a single symbol, which entails changes in the constants used in the theorems.

Let \mathbb{N} be the set of non-negative integers $\{0, 1, \dots\}$. For any string $\sigma \in \Sigma^*$, $|\sigma|$ is the length of σ . For $j \in \mathbb{N}$ such that $j \leq |\sigma|$, we'll use the notation $\sigma[j]$ to indicate the string that is left over after the first j symbols of σ are stripped off.

A Benenson automaton is parameterized by four numbers. Parameter n is the number of inputs that the automaton is sensitive to. Further, parameter S corresponds to the sticky end size, D to the maximum cutting range of the restriction enzyme (see Fig. 5.1(a)), and L to the length of the computational portion of the state molecule. A particular Benenson automaton is defined by specifying a state string σ and a selection of input-dependent cutting rules \mathcal{R} as follows.

Definition 5.3.1. *A Benenson automaton is a tuple $(S, D, L, \Sigma, n, \sigma, \mathcal{R})$ where $n, S, D, L \in \mathbb{N}$, Σ is a finite alphabet, $\sigma \in \Sigma^L$ is a state string and $\mathcal{R} \subseteq \{0, \dots, n\} \times \{0, 1\} \times \Sigma^S \times \{1, \dots, D\}$ is a rule set using sticky ends of length S and maximum cutting distance D . Each rule (i, b, ω, d) specifies an input i , a binary value b , a sticky end ω , and a cutting distance d .*

Interpreted as a DNA state molecule, $\sigma[j]$ represents the remaining portion of the molecule after j initial bases have been cut off. The first S symbols of $\sigma[j]$ represent the single-stranded sticky end overhang. This revealed sticky end ω and the value of an input bit x_i determine where the next cut will be made by the application of some cutting rule (i, b, ω, d) which is applicable if $x_i = b$ and cuts at a distance d .

Definition 5.3.2. *Given a Benenson automaton $(S, D, L, \Sigma, n, \sigma, \mathcal{R})$, for a binary input $x = x_1 x_2 \dots x_n$, a rule $(i, b, \omega, d) \in \mathcal{R}$ applies to $\sigma[j]$, where $j \in \mathbb{N}$ s.t. $|\sigma[j]| \geq S + d$, if $x_i = b$ and ω is the initial S symbol portion of $\sigma[j]$. We write $\sigma[j] \rightarrow_x \sigma[j + d]$ iff there exists a rule $(i, b, \omega, d) \in \mathcal{R}$ that applies to $\sigma[j]$. Further, \rightarrow_x^* is the reflexive transitive closure of \rightarrow_x .*

Our definition of Benenson automata (as well as the biochemical implementation) allows for conflicting cutting rules. For example, if the rule set contains rules $(1, 0, \omega, 4)$ and $(2, 1, \omega, 6)$, then either 4 or 6 bases may be cut off if the sticky end ω is revealed and $x_1 = 0, x_2 = 1$. An important class of Benenson automata are those in which it is impossible for conflicting cutting rules to apply simultaneously:

Definition 5.3.3. *A Benenson automaton $(S, D, L, \Sigma, n, \sigma, \mathcal{R})$ is said to be deterministic if $\forall x \in \{0, 1\}^n$ and $j \in \mathbb{N}$ s.t. $\sigma \rightarrow_x^* \sigma[j]$, there exists at most one $j' \in \mathbb{N}$ such that $\sigma[j] \rightarrow_x \sigma[j']$.*

While in computer science non-determinism often seems to increase computational power, we'll see this is not the case with Benenson automata. On the other hand, implementing deterministic Benenson automata

may be advantageous because (assuming error-free operation) each state molecule is cut up in the same way and thus there is no need for a combinatorially large number of state molecules.

Cutting the state string far enough indicates a 1 output. We'll think of Benenson automata computing Boolean functions as follows:

Definition 5.3.4. For $p \in \mathbb{N}$, we say that a Benenson automaton $(S, D, L, \Sigma, n, \sigma, \mathcal{R})$ non-deterministically computes a Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ at position p if $\forall x \in \{0, 1\}^n, f(x) = 1 \Leftrightarrow (\exists j \in \mathbb{N}, p \leq j \leq |\sigma| \text{ s.t. } \sigma \rightarrow_x^* \sigma[j])$. We'll say simply that the Benenson automaton non-deterministically computes f if such a p exists.

Definition 5.3.5. For $p \in \mathbb{N}$, we say that a Benenson automaton $(S, D, L, \Sigma, n, \sigma, \mathcal{R})$ computes a Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ at position p if the automaton is deterministic and $\forall x \in \{0, 1\}^n, f(x) = 1 \Leftrightarrow (\exists j \in \mathbb{N}, p \leq j \leq |\sigma| \text{ s.t. } \sigma \rightarrow_x^* \sigma[j])$. We'll say simply that the Benenson automaton computes f if such a p exists.

Other reasonable output conventions have the same computational power. For example, the following lemma shows that Benenson automata cutting to exactly p symbols to output a 1 and never cutting to exactly p symbols to indicate a 0, can be easily modified to output according to our convention.

Lemma 5.3.1. If for a deterministic Benenson automaton $(S, D, L, \Sigma, n, \sigma, \mathcal{R})$ and $f : \{0, 1\}^n \rightarrow \{0, 1\}$, $\exists p \in \mathbb{N}, p \leq L$ s.t. $\forall x \in \{0, 1\}^n, \sigma \rightarrow_x^* \sigma[p] \Leftrightarrow f(x) = 1$, then there is a Benenson automaton $(S, D, p + S, \Sigma, n, \sigma', \mathcal{R})$ that computes f .

An identical lemma also holds for non-deterministic computation. The lemma is trivially proven by taking σ' to be the first $p + S$ symbols of σ . All our constructions of Section 5.5 will produce Benenson automata requiring Lemma 5.3.1 to satisfy our definition of computing Boolean functions (Definition 5.3.5).

Note that interpreted as a DNA state molecule, the length of the remaining state string minus S represents the remaining double-stranded stem holding the output loop closed. Thus, as mentioned in the Introduction, automata from our constructions (like any automata produced by the above lemma) leave no stem only on a 1 output, allowing the loop to open.

In a biochemical implementation, it may seem that in order to change the input (say from being all zeros to all ones) it may be necessary to activate or inactivate a rule molecule for every cutting rule in \mathcal{R} . However, for certain Benenson automata much smaller changes need be made. Consider the example of an automaton whose rule set contains the rules $(1, 0, \omega, d)$ and $(1, 1, \omega, d)$. This pair of rules is really a single input-independent rule to cut d bases if sticky end ω is found no matter what the input is; thus, the cutting rule molecule for it can be always active in solution. The following definition quantifies the maximum "amount of effort" needed the change the input for a given Benenson automaton.

Definition 5.3.6. For $s \in \mathbb{N}$, a Benenson automaton $(S, D, L, \Sigma, n, \sigma, \mathcal{R})$ is said to be s -encoded if for every input bit i , $1 \leq i \leq n$, there are at most s sticky ends $\omega \in \Sigma^S$ such that $\exists(i, b, \omega, d) \in \mathcal{R}$ but $(i, 1 - b, \omega, d) \notin \mathcal{R}$.

An s -encoded automaton has at most s sticky ends "reading" any given input bit. In order to change the input, in a biochemical implementation of a deterministic s -encoded Benenson automaton, it is enough to activate or inactivate at most s pairs of rule molecules per changed bit.

5.4 Characterizing the Computational Power of Benenson Automata

In Section 5.5 we show that to compute function families using Benenson automata, only logarithmic scaling of the restriction enzyme sticky end size, and no scaling of the maximum cutting distance is needed. This result holds no matter what the complexity of the function family is. Further, if the family of functions is computable by log-depth circuits*, then a state string of only polynomial size is required. All of our constructions use deterministic Benenson automata.

*For the purposes of this paper, circuits are feed-forward and consist of AND, OR, and NOT gates with fan-in bounded by 2. For an introduction to circuit complexity see for example [9].

Theorem 5.4.1.

- (a) Any function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ can be computed by a Benenson automaton with sticky end size $S = O(\log n)$ and maximum cutting distance $D = O(1)$.
- (b) Families of functions computable by $O(\log n)$ depth circuits can be computed by Benenson automata with sticky end size $S = O(\log n)$, maximum cutting distance $D = O(1)$, and state string length $L = \text{poly}(n)$.

The constants implicit in both statements are rather small. (In this and in the following discussions we assume that the alphabet size $|\Sigma|$ is a constant.) Note that the sticky end size cannot be smaller than $O(\log n)$ since there must be at least a different sticky end for each input bit (otherwise the input isn't completely "read"). Thus, in computing arbitrary Boolean functions, we cannot do better than $S = O(\log n)$ and $D = O(1)$.

Further, in Section 5.6 we prove that our computation of families of functions computable by log-depth circuits is optimal, and neither allowing non-determinism nor larger sticky ends adds computational power:

Theorem 5.4.2. *Families of functions computable, possibly non-deterministically, by Benenson automata with $D = O(1)$, $L = \text{poly}(n)$ can be computed by $O(\log n)$ -depth circuits.*

Corollary 5.4.1. *Benenson automata with $S = O(\log n)$, $D = O(1)$, $L = \text{poly}(n)$ can compute the same class of families of functions as $O(\log n)$ -depth circuits.*

So if we consider only Benenson automata with $S = O(\log n)$, $D = O(1)$, $L = \text{poly}(n)$ efficient, then Benenson automata can compute a family of non-uniform functions efficiently if and only if it can be computed by a circuit of logarithmic depth. In Section 5.6, we'll also show that relaxing this notion of efficiency to include logarithmic cutting range does not increase the computational power significantly.

5.5 Simulating Branching Programs and Circuits

Benenson automata are closely related to the computational model known as branching programs. (For a review of branching programs see [12].) In the next section we show how arbitrary branching programs can be simulated. In the following two sections, we show how restricted classes of branching programs (fixed-width and permutation branching programs) can be simulated by Benenson automata with $S = O(\log n)$ and $D = O(1)$. Since fixed-width permutation branching programs are still powerful enough to compute arbitrary Boolean functions (Section 5.5.4), Theorem 5.4.1(a) follows. Further, in Section 5.5.4 we'll also see that fixed-width permutation branching programs of $\text{poly}(n)$ size can simulate $O(\log n)$ depth circuits, implying Theorem 5.4.1(b).

5.5.1 General Branching Programs

A branching program is a directed acyclic graph with three types of nodes: variable, accept and reject (e.g., Fig. 5.2(a)). The variable nodes are labeled with an input variable x_i ($1 \leq i \leq n$) and have two outgoing edges, one labeled 0 and the other 1, that lead to other (variable, accept, or reject) nodes. The accept and reject nodes have no outgoing edges. One variable node with no incoming edges is designated the start node. The process of computation consists of starting at the start node and at every node x_i , following the outgoing edge whose label matches the value of the i^{th} bit of the input. If an accept node is reached, we say that the branching program accepts the input x . Otherwise, a reject node is reached, and we say that the branching program rejects the input x . The function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ computed by a branching program is $f(x) = 1$ if x is accepted and 0 otherwise.

Because a branching program is a directed acyclic graph, we can index the nodes in such a way that we can never go from a node with a higher index to a node with a lower one (as shown in Fig. 5.2(a)). We can ensure that the first node is the start node and that there is only one accept node (convert all other accept nodes into variable nodes with all outgoing edges to this accept node). Let H be the total number of nodes in the given branching program. To each node with index $q \in \{1, \dots, H\}$ we associate a unique string

$\sigma_q \in \Sigma^*$ of length $S = \lceil \log_{|\Sigma|}(H) \rceil$. Let the state string σ be the concatenation of these segments in order: $\sigma_1 \dots \sigma_H$. Thus, the size L of the state string is HS . For every variable node q labeled x_i , define $\text{var}(q) = i$. Further, for every variable node q , $\text{goto}_0(q) \in \{q+1, \dots, H\}$ is the node targeted by the 0 outgoing edge and $\text{goto}_1(q) \in \{q+1, \dots, H\}$ is the node targeted by the 1 outgoing edge of q . Using this notation, the rule set of our automaton consists of the following cutting rules. For every variable node q , there are two rules: $(\text{var}(q), 0, \sigma_q, (\text{goto}_0(q) - q)S)$ and $(\text{var}(q), 1, \sigma_q, (\text{goto}_1(q) - q)S)$. Depending on the branching program, the cutting distance may have to be as large as $(H-1)S$ if $\text{goto}_0(1) = H$ or $\text{goto}_1(1) = H$.

By construction, for any remaining portion of the state string $\sigma_q \dots \sigma_{q'} \dots \sigma_H$, we have that $\sigma_q \dots \sigma_{q'} \dots \sigma_H \xrightarrow{x} \sigma_{q'} \dots \sigma_H$ iff the branching program goes to node q' from q on input x in one step. This implies that $\sigma_1 \dots \sigma_q \dots \sigma_H \xrightarrow{x}^* \sigma_q \dots \sigma_H$ iff the branching program eventually goes from the start node to node q on input x . Thus, this Benenson automaton cuts to the beginning of the segment corresponding to the accept node iff the branching program accepts the input x . Thus, employing Lemma 5.3.1 (i.e., shortening the state string) we have a Benenson automaton computing the function f computed by the branching program. As there is exactly one outgoing edge from any variable node for each value of the read input bit, it follows that the resultant automaton is deterministic. See Fig. 5.2(a,b) for an example of a branching program and the corresponding Benenson automaton. Thus we have the following lemma:

Lemma 5.5.1. *For any function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ computed by a branching program of H nodes and any alphabet Σ s.t. $|\Sigma| \geq 2$, there is a deterministic Benenson automaton $(S, D, L, \Sigma, n, \sigma, \mathcal{R})$ with sticky end size $S = \lceil \log_{|\Sigma|}(H) \rceil$, maximum cutting distance $D = (H-1)S$, and state string length $L \leq HS$ computing f .*

Note that all three complexity parameters (S , D , and L) of Benenson automata needed to simulate general branching programs using the above construction scale with the size of the branching program. Thus, for families of functions for which the size of branching programs computing them increases very fast with n , new restriction enzymes must be developed that scale similarly. Consequently, this is not enough to prove Theorem 5.4.1(a).

5.5.2 Fixed-Width Branching Programs

In this section, we demonstrate a sufficiently powerful subclass of branching programs whose simulation is possible by Benenson automata such that only the size L of the state string scales with the size of the branching program, while $S = O(\log n)$ and $D = O(1)$.

In the general case discussed in Section 5.5.1, our cutting range had to be large because we had no restriction on the connectivity of the branching program and may have needed to jump far. Further, we used a different sticky end for each node because there may be many different ‘‘connectivity patterns.’’ Restricting the connectivity of a branching program in a particular way permits optimizing the construction to significantly decrease S and D . In fact, both will lose their dependence on the size of the branching program. In the final construction, the sticky end size S will depend only on the size of the input n and the cutting range will be a constant.

A width J , length K branching program consists of K layers of J nodes each (e.g., Fig. 5.2(c)). The total number of nodes is $H = KJ$. We will think of J as a constant since for our purposes $J \leq 5$ will be enough. Nodes in each layer have outgoing edges only to the next layer, and every node in the last layer is either accepting or rejecting. We can ensure that the first node in the first layer is the start node and that the last layer has a single accept node. (Otherwise, the branching program can be trivially modified.) It turns out that width 5 branching programs are sufficiently powerful to simulate any circuit (Section 5.5.4). Further, the results of Section 5.6 ensure that we have not restricted our model of computation too much; more general Benenson automata cannot compute more efficiently.

Given a width J branching programs, we index nodes consecutively from each layer: the j th node in layer k obtains index $q = (k-1)J + j$. We use the same cutting rules as before, and construct the state string identically to the previous section, but with the following difference. Instead of using a unique segment for each node in the branching program as we did in the previous section, we let $\sigma_q = \sigma_{q'}$ iff $\text{var}(q) = \text{var}(q')$, $\text{goto}_0(q) - q = \text{goto}_0(q') - q'$ and $\text{goto}_1(q) - q = \text{goto}_1(q') - q'$. In other words, we allow the segments to be the same if their cutting rules have the same behavior. This does not change the behavior of the automaton but allows us to use fewer unique segments, thereby decreasing S . For a width J branching program, $\text{goto}_0(q) - q$

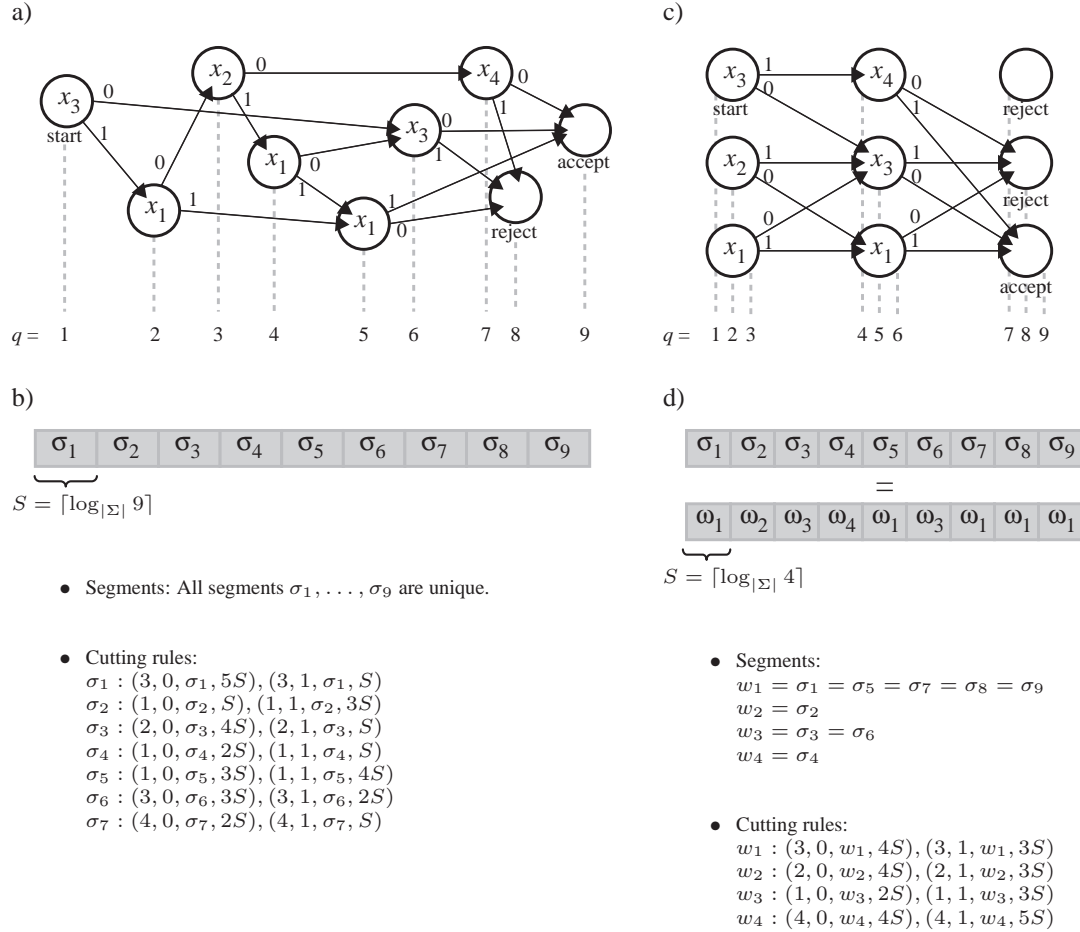


Figure 5.2: (a) An example of a general branching program of 9 nodes over 4 inputs and (b) the corresponding Benenson automaton. (c) An example of a width 3 branching program of 9 nodes over 4 inputs and (d) the corresponding Benenson automaton. Note that some nodes are inaccessible but these will be a small fraction for large programs. In both examples, $\sigma_1 \cdots \sigma_9 \xrightarrow{x} \sigma_9$ iff the branching program accepts x .

and $goto_1(q) - q$ range from 1 to $2J - 1$. So we need no more than $n(2J - 1)^2$ different segments, which implies that at most we need $S = \lceil \log_{|\Sigma|} (n(2J - 1)^2) \rceil$. (The segments corresponding to the accept and reject nodes can be anything as long as we cannot go from a reject node to the accept node. We can choose a segment such that $goto_0(q) - q, goto_1(q) - q \geq J$.) Note that the resultant automaton is $(2J - 1)^2$ -encoded as $goto_0(q) - q$ and $goto_1(q) - q$ range from 1 to $2J - 1$. Further, the maximum cutting distance needs to be at most $D = (2J - 1)S$ since in the worst case we need to go from the first node of a layer to the last node of the next layer. See Fig. 5.2(c,d) for an example of how a fixed-width branching program can be converted to a Benenson automaton.

As a result of the above optimizations for fixed-width branching programs, the sticky end size S and the maximum cutting distance D lose their dependence on the length of the branching program K . Assuming the width J is fixed, this means that the choice of the restriction enzyme is independent of the size of the branching program and is dependent only on the number of input bits n .

Lemma 5.5.2. *For any function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ computed by a branching program of width J and length K , and any alphabet Σ s.t. $|\Sigma| \geq 2$, there is a $(2J - 1)^2$ -encoded deterministic Benenson automaton $(S, D, L, \Sigma, n, \sigma, \mathcal{R})$ with sticky end size $S = \lceil \log_{|\Sigma|} (n(2J - 1)^2) \rceil$, maximum cutting distance $D = (2J - 1)S$, and state string length $L \leq KJS$ computing f .*

The constructions described above rely on being able to skip entire segments in a single cut. It seems that the cutting range must be at least logarithmic in n , since the size of the segments is logarithmic in n to be able to read all the input variables. However, in the following we describe the construction in which the maximum cutting distance D is dependent only on the width J and no longer on n , and is thus shorter than the segments. As before, we will still have that $\sigma = \sigma_1 \cdots \sigma_q \cdots \sigma_H \xrightarrow{x}^* \sigma_q \cdots \sigma_H$ iff the branching program eventually goes from the start node to node q on input x . However, while previously following a single arrow on the branching program corresponded to the application of a single cutting rule, now it will involve the application of many. We'll separate the cutting rules into two logical types: *segment* cutting rules and *skip* cutting rules. If previously the applicable cutting rule removed $(goto_0(q) - q)$ or $(goto_1(q) - q)$ entire segments, now the corresponding segment cutting rule only removes $(goto_0(q) - q)$ or $(goto_1(q) - q)$ symbols from the beginning of the current segment σ_q . How can the cutting of d symbols from the beginning of a segment result in the eventual cutting of d entire segments? This is accomplished by the skip cutting rules as follows (see also Fig. 5.3).

A new symbol $\iota \in \Sigma$ marks the beginning of each segment, while the rest of the segment uses symbols in $\Sigma - \{\iota\}$. A skip cutting rule is always applicable if the first symbol of the revealed sticky end is not ι , while segment cutting rules are only applicable if the first symbol of the revealed sticky end is ι . All skip cutting rules cut exactly D symbols. We use segments of length $m = D \cdot k + 1$ for some integer $k \geq 1$. After the application of some segment cutting rule removes d initial symbols of the state string, exactly $d \cdot k$ applications of skip cutting rules follow because after $d \cdot k \cdot D + d = d \cdot m$ symbols have been removed, it follows that d entire segments (each of length m) have been cut off and a new segment cutting rule is applicable. No segment cutting rule is applicable before then since this is the first time the first symbol of the revealed sticky end is ι .

Formally, we use segments of the form $\sigma_q = \iota\tau_q\nu$, where $\tau_q, \nu \in (\Sigma - \{\iota\})^*$ and ν is an arbitrary string such that $|\sigma_q| = D \cdot k + 1$ for some integer $k \geq 1$. The strings τ_q are chosen such that $\tau_q = \tau_{q'}$ iff $var(q) = var(q')$, $goto_0(q) - q = goto_0(q') - q'$, and $goto_1(q) - q = goto_1(q') - q'$. For each variable node q , the segment cutting rules are: $(var(q), 0, \iota\tau_q, (goto_0(q) - q))$ and $(var(q), 1, \iota\tau_q, (goto_1(q) - q))$. Since we have at most $n(2J - 1)^2$ unique τ_q s and we also need to read the ι , we need the sticky end to be of size $S = 1 + \lceil \log_{|\Sigma|-1} (n(2J - 1)^2) \rceil$. In the worst case, as before, we have $goto_{0/1}(q) - q = 2J - 1$ and so the maximum cutting distance needs to be $D = 2J - 1$ so that we can skip $2J - 1$ segments. Since the skip cutting rules should be independent of the input, for every $\omega \in \Sigma^S$ s.t. the first symbol of ω is not ι , we can use the following two rules: $(1, 0, \omega, D)$ and $(1, 1, \omega, D)$. Note that since for both segment and skip cutting rules, there is at most one cutting rule applicable at any time, and because a segment cutting rule cannot be applicable at the same time as a skip cutting rule, it follows that our construction yields a deterministic Benenson automaton.

With the above trick (of course after applying Lemma 5.3.1), we have the following lemma for fixed-width branching programs:

Lemma 5.5.3. *For any function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ computed by a branching program of width J and length K , and any alphabet Σ s.t. $|\Sigma| \geq 3$, there is a $(2J - 1)^2$ -encoded deterministic Benenson automaton $(S, D, L, \Sigma, n, \sigma, \mathcal{R})$ with sticky end size $S = 1 + \lceil \log_{|\Sigma|-1} (n(2J - 1)^2) \rceil$, maximum cutting distance $D = 2J - 1$, and state string length $L \leq KJS$ computing f .*

Lemma 5.5.3 together with Barrington's theorem (Lemma 5.5.5) is enough to prove both parts of Theorem 5.4.1. However, we first optimize our construction even further to obtain better constants.

5.5.3 Permutation Branching Programs

We can obtain better constants if we restrict the branching program even more. Again, in the next section we'll see that, even with this restriction, branching programs can simulate circuits.

First, we need a notation for the context of layered branching programs. For node j in layer k let $goto_0(k, j) = j'$ if the j' th node in layer $k + 1$ is targeted by the 0 outgoing edge of this node; $goto_1(k, j)$ is defined analogously. A width J permutation branching program is a width J branching program such that for all layers k , the sequences $goto_0(k, 1), \dots, goto_0(k, J)$ and $goto_1(k, 1), \dots, goto_1(k, J)$ are permutations of $1, \dots, J$. Further, there is exactly one accept node in the last layer (this can no longer be trivially assumed). It turns out that width 5 permutation branching programs are still sufficiently powerful to simulate any circuit

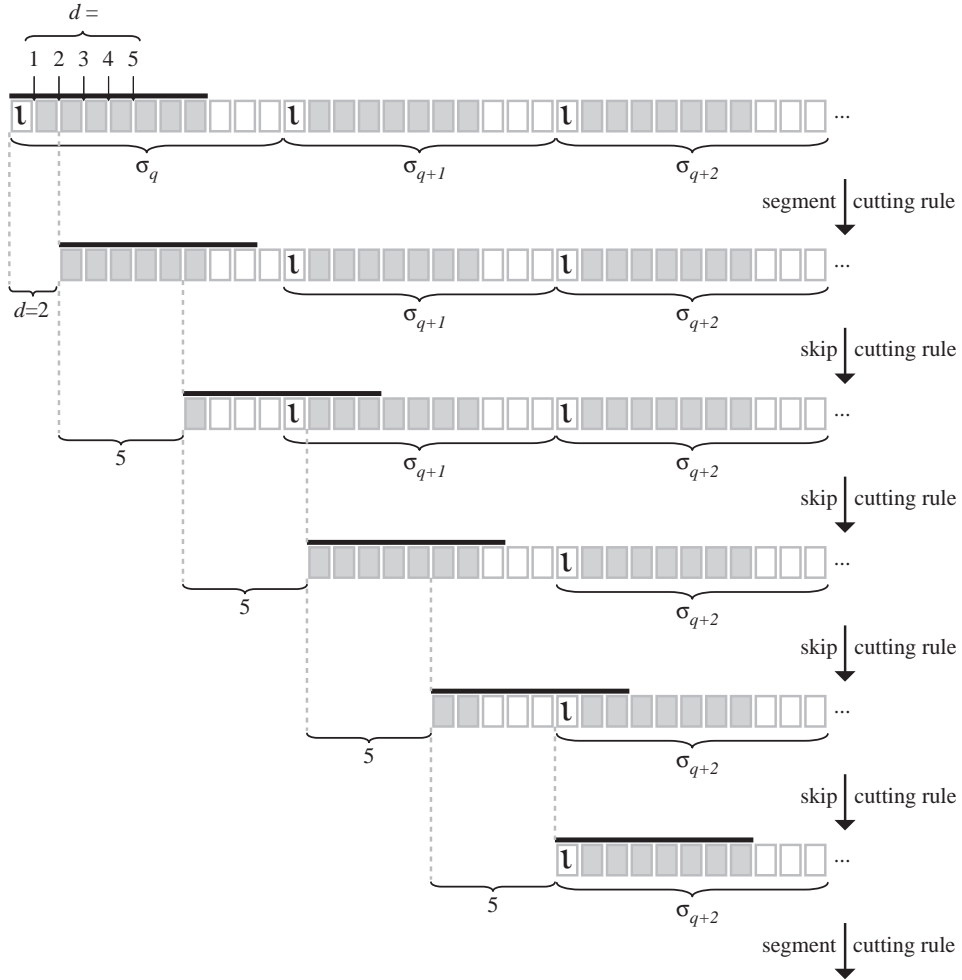


Figure 5.3: An example of a segment cutting rule application and the subsequent application of skip cutting rules. In this case, $D = 5$, $k = 2$, and the size of the segments is $m = D \cdot k + 1 = 11$. The sticky end size is $S = 8$; the black horizontal lines above the state string show the sticky end in each step. The grayed squares comprise τ_q , τ_{q+1} , and τ_{q+2} that, together with a bit of input, determine which segment cutting rule is applicable. The empty white squares comprise ν .

(Section 5.5.4). In Section 5.6, we'll confirm that we have not restricted our model of computation too much: efficient Benenson automata cannot simulate anything more powerful than permutation branching programs.

For permutation branching programs we can use fewer unique sequences for the τ_q s than for general fixed width branching programs. It is easy to see that for every permutation branching program, there is another permutation branching program of the same width and length that accepts the same inputs as the original program but for all layers k , $goto_0(k, \cdot)$ is the identity permutation (i.e., $goto_0(k, j) = j$). In this case, since $goto_0(q) - q$ is always J , we need at most $n(2J - 1)$ unique τ_q s. Thus, sticky ends of size $S = 1 + \lceil \log_{|\Sigma|-1}(n(2J - 1)) \rceil$ are sufficient and our automaton is $(2J - 1)$ -encoded. This leads to the following lemma:

Lemma 5.5.4. *For any function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ computed by a permutation branching program of width J and length K , and any alphabet Σ s.t. $|\Sigma| \geq 3$, there is a $(2J - 1)$ -encoded deterministic Benenson automaton $(S, D, L, \Sigma, n, \sigma, \mathcal{R})$ with sticky end size $S = 1 + \lceil \log_{|\Sigma|-1}(n(2J - 1)) \rceil$, maximum cutting distance $D = 2J - 1$, and state string length $L \leq KJS$ computing f .*

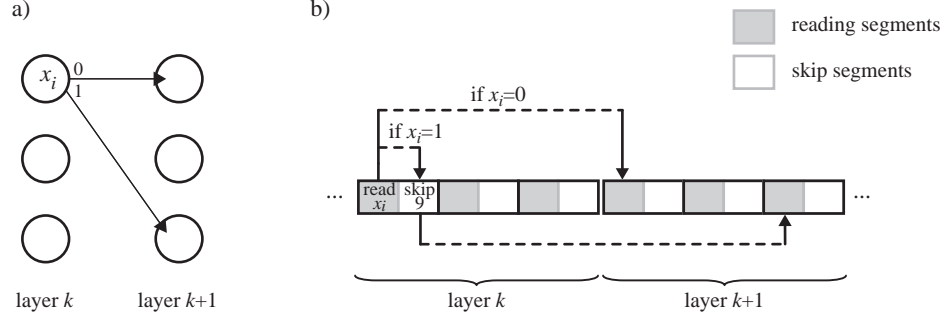


Figure 5.4: Illustration of the construction achieving 1-encoded automata. (a) The portion of the branching program being simulated. In this case the width of the branching program is $J = 3$. (b) The relevant portion of the Benenson automaton. Note that each skip illustrated by the dashed lines consists of many cuts like those illustrated in Fig. 5.3.

5.5.4 Simulating Circuits

While it may seem that fixed-width permutation branching programs are a very weak model of computation, it turns out that to simulate circuits, width 5 permutation branching programs is all we need:

Lemma 5.5.5 (Barrington [2]). *A function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ computed by a circuit of depth C can be computed by a length 4^C width 5 permutation branching program.*

Corollary 5.5.1 (of Lemmas 5.5.4 and 5.5.5). *For any function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ computed by a circuit of depth C , and any alphabet Σ s.t. $|\Sigma| \geq 3$, there is a 9-encoded deterministic Benenson automaton $(S, D, L, \Sigma, n, \sigma, \mathcal{R})$ with sticky end size $S = 1 + \lceil \log_{|\Sigma|-1} (9n) \rceil$, maximum cutting distance $D = 9$, and state string length $L = 4^C 5S$ computing f .*

This provides an alternative proof of Theorem 5.4.1 and implies, for instance, that a Benenson automaton using the restriction enzyme *FokI* can do arbitrary 3-bit computation. Any increase in the sticky end size, exponentially increases the number of inputs that can be handled. If an enzyme is discovered that cuts 9 bases away like *FokI* but leaves size 7 sticky ends, then it can do all 81-bit computation.

Letting $C = O(\log n)$, this proves Theorem 5.4.1(b). Theorem 5.4.1(a), of course, follows trivially since the complexity of the circuit (depth C) enters only in the length of the state string.

5.5.5 Achieving 1-Encoded Automata

If it is essential that the Benenson automaton be 1-encoded, the scheme from Section 5.5.3 can be adapted at the expense of slightly increasing the maximum cutting range D and the length of the state string L . The modification actually decreases the size of the sticky ends.

We provide a sketch of the construction; the details are carried over from the previous sections. The main idea is to use a pair of segments $\sigma_q = \nu\tau_q$ and $\sigma'_q = \nu\tau'_q$, where $\tau_q, \tau'_q \in (\Sigma - \{\iota\})^*$, for each node q of the permutation branching program, rather than a single segment as before (see Fig. 5.4). The first segment of the pair σ_q (the *reading segment*) reads the corresponding variable and skips either $2J$ segments if $x_i = 0$ or goes to the next segment if $x_i = 1$. Thus, the segment cutting rules for this segment are: $(i, 0, \nu\tau_q, 2J)$ and $(i, 1, \nu\tau_q, 1)$. Segment σ'_q (the *skip segment*) encodes an input-independent skip of $2(\text{goto}_1(q) - q) - 1$ segments to go to the correct reading segment. Thus, for the skip segment we can use the following segment cutting rules: $(1, 0, \nu\tau'_q, 2(\text{goto}_1(q) - q) - 1)$ and $(1, 1, \nu\tau'_q, 2(\text{goto}_1(q) - q) - 1)$. We need at most $n + 2J - 1$ unique segment types: n to read all the variables, and $2J - 1$ to be able to skip $2(\text{goto}_1(q) - q) - 1$ segments for all the values of $(\text{goto}_1(q) - q)$ which ranges from 1 to $2J - 1$. The maximum number of segments to skip is $2(2J - 1) - 1 = 4J - 3$. Note that there is at most one reading segment per input bit and thus the construction is 1-encoded.

Lemma 5.5.6. *For any function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ computed by a permutation branching program of width J and length K , and any alphabet Σ s.t. $|\Sigma| \geq 3$, there is 1-encoded deterministic Benenson automaton $(S, D, L, \Sigma, n, \sigma, \mathcal{R})$ with $S = 1 + \lceil \log_{|\Sigma|-1} (n + 2J - 1) \rceil$, $D = 4J - 3$, and $L \leq 2KJS$ computing f .*

This implies, for instance, that 1-encoded Benenson automata using restriction enzyme *FokI* can simulate any width 3 permutation branching program over 22 inputs.

Corollary 5.5.2 (of Lemmas 5.5.6 and 5.5.5). *For any function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ computed by a circuit of depth C , and any alphabet Σ s.t. $|\Sigma| \geq 3$, there is a 1-encoded deterministic Benenson automaton $(S, D, L, \Sigma, n, \sigma, \mathcal{R})$ with $S = 1 + \lceil \log_{|\Sigma|-1} (n + 9) \rceil$, $D = 17$, and $L = 4^C 10S$ computing f .*

This implies, for example, that if a DNA restriction enzyme can be found that leaves sticky ends of size 4 like *FokI* but cuts 17 bases away, then this enzyme can do all 18 bit computation with 1-encoded Benenson automata.

5.6 Shallow Circuits to Simulate Benenson automata

We'll now show that our constructions from the previous section are asymptotically optimal.

Lemma 5.6.1. *A function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ computed, possibly non-deterministically, by a Benenson automaton $(S, D, L, \Sigma, n, \sigma, \mathcal{R})$ can be computed by a $O(\log(L/D) \log D + D)$ depth, $O(D^4 L)$ size circuit.*

To see that this lemma implies Theorem 5.4.2, take $D = O(1)$, $S = O(\log n)$, and $L = \text{poly}(n)$. Further, this Lemma implies that allowing non-determinism does not increase the computational power of Benenson automata. Likewise, note that sticky end size S does not affect the complexity of the circuit simulating a Benenson automaton. This implies that increasing the sticky end size to be larger than $O(\log n)$ does not increase computational power.

Finally, Lemma 5.6.1 implies that Benenson automata using maximum cutting distance $D = O(\log n)$, and state string length $L = \text{poly}(n)$ cannot be much more powerful than Benenson automata with $D = O(1)$, and $L = \text{poly}(n)$. Specifically, $\forall \varepsilon > 0$, functions computable by Benenson automata with $D = O(\log n)$, and $L = \text{poly}(n)$ are computable by $O(\log^{1+\varepsilon} n)$ depth, $\text{poly}(n)$ size circuits.

Let us be given a Benenson automaton $(S, D, L, \Sigma, n, \sigma, \mathcal{R})$ computing, possibly non-deterministically, a Boolean function f at position p . Observe that in order to check if, for a given input, the state string can be cut to or beyond p , it is enough to check if it can be cut to p or the following D symbols. The idea of our construction is that we split the state string into segments of length D and compute for all cut locations in every segment where the possible cuts in the next segment can be (for the given input). Then this information can be composed using a binary tree of matrix multiplications to reveal all possible cuts in the D symbols following p starting with the full state string. Making the segments shorter than D allows the possibility that a cut entirely bypasses a segment, thereby fouling the composition, and making them longer than D makes the construction less efficient (i.e., results in a deeper circuit). This proof is similar to the argument that poly-length fixed-width branching programs can be simulated by log-depth circuits (e.g., [2]), in which the construction computes a binary tree of compositions of permutations rather than matrix multiplications.

For convenience let us assume p is divisible by D (say $Q = p/D$) and that $|\sigma| \geq p + D$. For q and $q' \in \mathbb{N}$, $q < q' \leq Q$, define a $D \times D$ binary matrix $T_{q,q'}(x)$ in which the h^{th} bit (0 indexed) of the j^{th} row (0 indexed) is 1 iff $\sigma[qD + j] \rightarrow_x^* \sigma[q'D + h]$. Observe that $T_{q,q'}(x) \times T_{q',q''}(x) = T_{q,q''}(x)$ where in the matrix multiplication $+$ is logical OR and \cdot is logical AND. Therefore, $f(x) = 1$ iff there is at least one 1 in the 0^{th} row of $T_{1,Q}(x) = T_{1,2}(x) \times T_{2,3}(x) \times \cdots \times T_{Q-1,Q}(x)$. If p is not divisible by D or $|\sigma| < p + D$, we can let the first or last of these matrices be smaller as necessary.

We can create a shallow binary tree computing the product $T_{1,Q}$. For clarity of exposition, let us assume that $Q - 1$ is a power of 2. Our circuit consists of gadgets A_q ($1 \leq q \leq Q - 1$), gadgets B , and gadget C (see Fig. 5.5). The input and output lines of gadgets B represent a matrix $T_{q,q'}(x)$ for a range of segments q to q' as shown in Fig. 5.5. To compute the initial series of matrices, each gadget A_q needs only to know at most $2D$ bits of input x on which $T_{q,q+1}(x)$ may depend (a segment of length D can read at most D input bits). Each gadget A_q can be a selector circuit that uses the relevant input bits to select one of 2^{2D} possible hardwired outputs (different for each A_q). These gadgets A_q have depth $O(D)$ and size $O(D^2 4^D)$. The

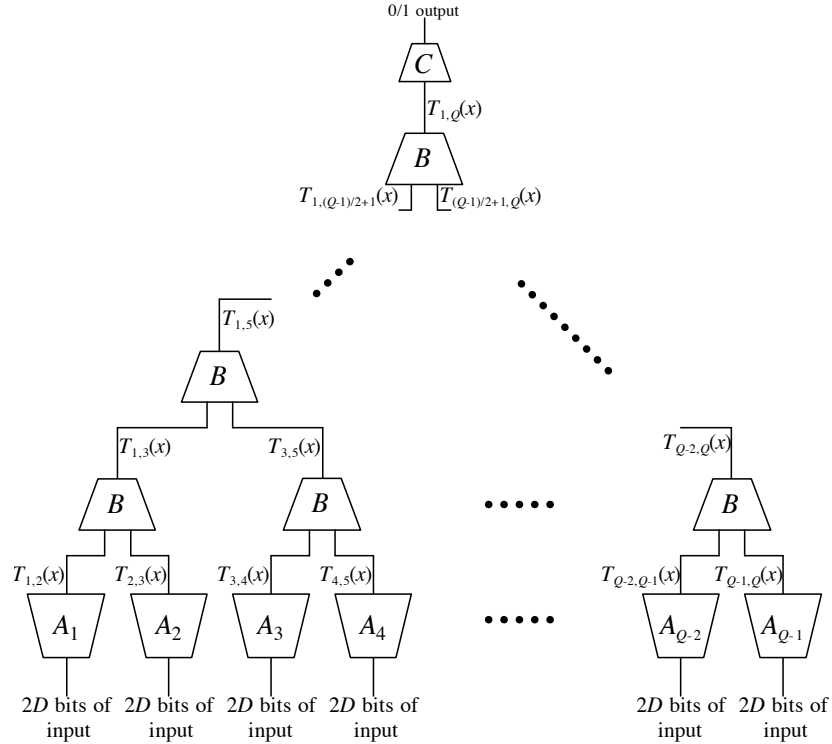


Figure 5.5: Circuit outline for simulating a Benenson automaton. The T lines represent a bundle of at most D^2 wires. Input lines represent a bundle of at most $2D$ wires (a different subset for each gadget, possibly overlapping).

output of gadget B is the product of its first input matrix by the second input matrix, where $+$ is logical OR and \cdot is logical AND. Gadget B can be made of depth $O(\log D)$ and size $O(D^3)$. Gadget C outputs 1 iff there is at least one 1 in the 0^{th} row of its input matrix.

5.7 Discussion

This work generalizes the non-uniform model of computation based on the work of Benenson et al. [4] and characterizes its computational power. We considered restriction enzymes with variable reach and sticky end size, and studied how the complexity of the possible computation scales with these parameters. In particular, we showed that Benenson automata can simulate arbitrary circuits and that polynomial length Benenson automata with constant cutting range are equivalent to fixed-width branching programs and therefore equivalent to log-depth circuits. We achieve these asymptotic results with good constants, suggesting that the insights and constructions developed here may have applications.

There may be ways to reduce the constants in our results even further. Although the fixed-width permutation branching programs produced via Barrington's theorem have the same variable read by every node in a layer, this fact is not used in our constructions. Exploiting it may achieve smaller sticky end size or maximum cutting distance.

As mentioned in the Introduction, in a biochemical implementation of our constructions the last possible cut in the case that $f(x) = 0$ may have to be sufficiently far away from the output loop to prevent its erroneous opening. By using a few extra unique sticky ends we can achieve this with our constructions. For example, by adding one more unique sticky end corresponding to the reject states and making sure the accept state is last, we can ensure that in the constructions simulating general branching programs and fixed-width branching programs the last possible cut in the case $f(x) = 0$ is at least the length of a segment away ($\geq S, D$) from

the last cut in the case $f(x) = 1$.

Some Benenson automata may pose practical problems for existing or future restriction enzymes not discussed in this paper. For example, a cutting rule with $d = 1$ would require a single base adjacent to a nick to be cleaved off each strand, which may not be biochemically plausible for certain restriction enzymes (a ligation enzyme may have to be used). Such issues must be considered carefully for an experimental implementation.

The major problem with directly implementing our construction is the potential of an error during the attachment of the rule molecule and during cuts. While a practical implementation of a Benenson automaton [4] has to work reliably despite high error rates, our formalization does not take the possibility of erroneous cutting into account. Further work is needed to formalize and characterize effective error-robust computation with Benenson automata. Similarly, while it is easiest to study the binary model in which a reaction either occurs or not, a model of analog concentration comparisons may better match some types of tests implemented by Benenson et al.

Acknowledgments

We thank Georg Seelig for first bringing Benenson et al.'s work to our attention. Further, we thank two anonymous reviewers for very detailed reading of this paper and useful suggestions. This research was supported by NIH training grant MH19138-15.

Bibliography

- [1] R. Adar, Y. Benenson, G. Linshiz, A. Rosner, N. Tishby, and E. Shapiro. Stochastic computing with biomolecular automata. *Proceedings of the National Academy of Sciences*, 101:9960–9965, 2004.
- [2] D. A. Barrington. Bounded-width polynomial-sized branching programs recognize exactly those languages in NC¹. *Journal of Computer Systems Science*, 38:150–164, 1988.
- [3] Y. Benenson, R. Adar, T. Paz-Elizur, Z. Livneh, and E. Shapiro. DNA molecule provides a computing machine with both data and fuel. *Proceedings of the National Academy of Sciences*, 100:2191–2196, 2003.
- [4] Y. Benenson, B. Gil, U. Ben-dor, R. Adar, and E. Shapiro. An autonomous molecular computer for logical control of gene expression. *Nature*, 429:423–429, 2004.
- [5] Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, and E. Shapiro. Programmable and autonomous computing machine made of biomolecules. *Nature*, 414:430–434, 2001.
- [6] P. W. K. Rothmund. A DNA and restriction enzyme implementation of turing machines. In *DNA-Based Computers*, volume 27 of *Proceedings of a DIMACS Workshop*, pages 75–119, 1995.
- [7] T. Inui, H. Ikeda, and Y. Nakamura. The design of an artificial restriction enzyme having simultaneous DNA cleavage activity. *Nucleic Acids Symposium Series*, 44:141–142, 2000.
- [8] M. Komiyama. DNA manipulation using artificial restriction enzymes. *Tanpakushitsu Kakusan Koso*, 50:81–86, 2005. (in Japanese).
- [9] C. H. Papadimitriou. *Elements of the theory of computation*. Prentice-Hall, 1997.
- [10] M. N. Stojanovic, T. E. Mitchell, and D. Stefanovic. Deoxyribozyme-based logic gates. *Journal of the American Chemical Society*, 124:3555–3561, 2002.
- [11] H. Sugisaki and S. Kanazawa. New restriction endonucleases from *Flavobacterium okeanokoites* (FokI) and *Micrococcus luteus* (MluI). *Gene*, 16:73–78, 1981.
- [12] I. Wegener. *Branching Programs and Binary Decision Diagrams*. Society for Industrial and Applied Mathematics, 2000.

Chapter 6

Computation with Finite Stochastic Chemical Reaction Networks

Collaborators: Matthew Cook, Erik Winfree, and Jehoshua Bruck. **My contribution:** MC, EW, and I proved basic Turing-universality. I developed the fast construction, and wrote the text of the paper.

This chapter was published as: David Soloveichik, Matt Cook, Erik Winfree, and Shuki Bruck, "Computation with Finite Stochastic Chemical Reaction Networks," *Natural Computing* (on-line Feb 2008).

6.1 Abstract

A highly desired part of the synthetic biology toolbox is an embedded chemical microcontroller, capable of autonomously following a logic program specified by a set of instructions, and interacting with its cellular environment. Strategies for incorporating logic in aqueous chemistry have focused primarily on implementing components, such as logic gates, that are composed into larger circuits, with each logic gate in the circuit corresponding to one or more molecular species. With this paradigm, designing and producing new molecular species is necessary to perform larger computations. An alternative approach begins by noticing that chemical systems on the small scale are fundamentally discrete and stochastic. In particular, the exact molecular counts of each molecular species present, is an intrinsically available form of information. This might appear to be a very weak form of information, perhaps quite difficult for computations to utilize. Indeed, it has been shown that error-free Turing universal computation is impossible in this setting. Nevertheless, we show a design of a chemical computer that achieves fast and reliable Turing universal computation using molecular counts. Our scheme uses only a small number of different molecular species to do computation of arbitrary complexity. The total probability of error of the computation can be made arbitrarily small (but not zero) by adjusting the initial molecular counts of certain species. While physical implementations would be difficult, these results demonstrate that molecular counts can be a useful form of information for small molecular systems such as those operating within cellular environments.

6.2 Introduction

Many ways to perform computation in a chemical system have been explored in the literature, both as theoretical proposals and as practical implementations. The most common and, at present, successful attempts involve simulating Boolean circuits [36, 8, 34, 32]. In such cases, information is generally encoded in the high or low concentrations of various signaling molecules. Since each binary variable used during the computation requires its own signaling molecule, this makes creating large circuits onerous. Computation has also been suggested via a Turing machine (TM) simulation on a polymer [5, 30], via cellular automaton simulation in self-assembly [31], or via compartmentalization of molecules into membrane compartments [6, 29]. These approaches rely on the geometrical arrangement of a fixed set of parts to encode information. This allows unbounded computation to be performed by molecular systems containing only a limited set of types

of enzyme and basic information-carrying molecular components. It had been widely assumed, but never proven, that these two paradigms encompassed all ways to do computation in chemistry: either the spatial arrangement and geometric structure of molecules is used, so that an arbitrary amount of information can be stored and manipulated, allowing Turing universal computation; or a finite number of molecular species react in a well-mixed solution, so that each Boolean signal is carried by the concentration of a dedicated species, and only finite circuit computations can be performed.

Here we show that this assumption is incorrect: well-mixed finite stochastic chemical reaction networks with a fixed number of species can perform Turing universal computation with an arbitrarily low error probability. This result illuminates the computational power of stochastic chemical kinetics: error-free Turing universal computation is provably impossible, but once any non-zero probability of error is allowed, no matter how small, stochastic chemical reaction networks become Turing universal. This dichotomy implies that the question of whether a stochastic chemical system *can* eventually reach a certain state is always decidable, the question of whether this is *likely* to occur is uncomputable in general.

To achieve Turing-universality, a system must not require a priori knowledge of how long the computation will be, or how much information will need to be stored. For instance, a system that maintains some fixed error probability per computational step cannot be Turing universal because, after sufficiently many steps, the total error probability will become large enough to invalidate the computation. We avoid this problem by devising a reaction scheme in which the probability of error, according to stochastic chemical kinetics, is reduced at each step indefinitely. While the chance of error cannot be eliminated, it does not grow arbitrarily large with the length of the computation, and can in fact be made arbitrarily small without modifying any of the reactions but simply by increasing the initial molecular count of an “accuracy” species.

We view stochastic chemical kinetics as a model of computation in which information is stored and processed in the integer counts of molecules in a well-mixed solution, as discussed in [22] and [2] (see Section 6.6 for a comparison with our results). This type of information storage is effectively unary and thus it may seem inappropriate for fast computation. It is thus surprising that our construction achieves computation speed only polynomially slower than that achievable by physical processes making use of spatial and geometrical structure. The total molecular count necessarily scales exponentially with the memory requirements of the entire computation. This is unavoidable if the memory requirements are allowed to grow while the number of species is bounded. However, for many problems of interest memory requirements may be small. Further, our scheme may be appropriate for certain problems naturally conceived as manipulation of molecular “counts,” and may allow the implementation of such algorithms more directly than previously proposed. Likewise, engineering exquisite sensitivity of a cell to the environment may effectively require determining the exact intracellular molecular counts of the detectable species. Finally, it is possible that some natural processes can be better understood in terms of manipulating molecular counts as opposed to the prevailing regulatory circuits view.

The exponential trend in the complexity of engineered biochemical systems suggests that reaction networks on the scale of our construction may soon become feasible. The state of the art in synthetic biology progressed from the coupling of 2–3 genes in 2000 [9], to the implementation of over 100 deoxyribonuclease logic gates in vitro in 2006 [23]. Our construction is sufficiently simple that significant aspects of it may be implemented with the technology of synthetic biology of the near future.

6.3 Stochastic Model of Chemical Kinetics

The stochastic chemical reaction network (SCRN) model of chemical kinetics describes interactions involving integer number of molecules as Markov jump processes [26, 38, 11, 15]. It is used in domains where the traditional model of deterministic continuous mass action kinetics is invalid due to small molecular counts. When all molecular counts are large the model scales to the mass action law [20, 12]. Small molecular counts are prevalent in biology: for example, over 80% of the genes in the *E. coli* chromosome are expressed at fewer than a hundred copies per cell, with some key control factors present in quantities under a dozen [17, 21]. Experimental observations and computer simulations have confirmed that stochastic effects can be physiologically significant [25, 10, 37]. Consequently, the stochastic model is widely employed for modeling cellular processes (e.g., [3]) and is included in numerous software packages [35, 39, 19, 1]. The algorithms used for modeling stochastic kinetics are usually based on Gillespie’s algorithm [14, 13, 16].

Consider a solution containing p species. Its state is a vector $\mathbf{z} \in \mathbb{N}^p$ (where $\mathbb{N} = \{0, 1, 2, \dots\}$) specifying the integral molecular counts of the species. A reaction α is a tuple $\langle \mathbf{l}, \mathbf{r}, k \rangle \in \mathbb{N}^p \times \mathbb{N}^p \times \mathbb{R}^+$ which specifies the stoichiometry of the reactants and products, and the rate constant k . We use capital letters to refer to the various species and standard chemical notation to describe a reaction (e.g., $A + C \xrightarrow{k} A + 2B$). We write $\#_{\mathbf{z}}X$ to indicate the number of molecules of species X in state \mathbf{z} , omitting the subscript when the state is obvious. A *SCRN* \mathcal{C} is a finite set of reactions. In state \mathbf{z} a reaction α is possible if there are enough reactant molecules: $\forall i, \mathbf{z}_i - \mathbf{l}_i \geq 0$. The result of reaction α occurring in state \mathbf{z} is to move the system into state $\mathbf{z} - \mathbf{l} + \mathbf{r}$. Given a fixed volume v and current state \mathbf{z} , the propensity of a unimolecular reaction $\alpha : X_i \xrightarrow{k} \dots$ is $\rho(\mathbf{z}, \alpha) = k\#_{\mathbf{z}}X_i$. The propensity of a bimolecular reaction $\alpha : X_i + X_j \xrightarrow{k} \dots$ where $X_i \neq X_j$ is $\rho(\mathbf{z}, \alpha) = k\frac{\#_{\mathbf{z}}X_i\#_{\mathbf{z}}X_j}{v}$. The propensity of a bimolecular reaction $\alpha : 2X_i \xrightarrow{k} \dots$ is $\rho(\mathbf{z}, \alpha) = \frac{k}{2}\frac{\#_{\mathbf{z}}X_i(\#_{\mathbf{z}}X_i-1)}{v}$. Sometimes the model is extended to higher-order reactions [38], but the merit of this is a matter of some controversy. We follow Gillespie and others and allow unary and bimolecular reactions only. The propensity function determines the kinetics of the system as follows. If the system is in state \mathbf{z} , no further reactions are possible if $\forall \alpha \in \mathcal{C}, \rho(\mathbf{z}, \alpha) = 0$. Otherwise, the time until the next reaction occurs is an exponential random variable with rate $\sum_{\alpha \in \mathcal{C}} \rho(\mathbf{z}, \alpha)$. The probability that next reaction will be a particular α_{next} is $\rho(\mathbf{z}, \alpha_{next}) / \sum_{\alpha \in \mathcal{C}} \rho(\mathbf{z}, \alpha)$.

While the model may be used to describe elementary chemical reactions, it is often used to specify higher-level processes such as phosphorylation cascades, transcription, and genetic regulatory cascades, where complex multistep processes are approximated as single-step reactions. Molecules carrying mass and energy are assumed to be in abundant supply and are not modeled explicitly. This is the sense in which we use the model here because we allow reactions violating the conservation of energy and mass. While we will not find “atomic” reactions satisfying our proposed SCRN, a reasonable approximation may be attained with complex organic molecules, assuming an implicit source of energy and raw materials. The existence of a formal SCRNs with the given properties strongly suggests the existence of a real chemical system with the same properties. Thus, in order to implement various computations in real chemistry, first we should be able to write down a set of chemical reactions (a SCRNs), and then find a set of physical molecular species that behave accordingly. This approach is compatible with the philosophy of synthetic biology [32, 34]. Here we focus on the first step, reaction network design, and explore computation in SCRN assuming arbitrary reactions can be used, and that they behave according to the above model of stochastic kinetics.

6.4 Time/Space-Bounded Algorithms

There is a rich literature on abstract models of computation that make use of integer counts, primarily because these are among the simplest Turing universal machines known. Minsky’s register machine (RM) [27] is the prototypical example. A RM is a finite state machine augmented with fixed number of registers that can each hold an arbitrary non-negative integer. An $inc(i, r, j)$ instruction specifies that when in state i , increment register r by 1, and move to state j . A $dec(i, r, j, k)$ instruction specifies that when in state i , decrement register r by 1 if it is nonzero and move to state j ; otherwise, move to state k . There are two special states: start and halt. Computation initiates in the start state with the input count encoded in an input register, and the computation stops if the halt state is reached. The output is then taken to be encoded in the register values (e.g., the final value of the input register). While it may seem that a RM is a very weak model of computation, it is known that even two-register RMs are Turing universal. Given any RM, our task is to come up with a SCRNs that performs the same computation step by step. This SCRNs is then said to simulate the RM.

For a given RM, we may construct a simple SCRNs that simulates it with high probability as follows. We use a set of state species $\{S_i\}$, one for each state i of the RM, and set of register species $\{M_r\}$, one for each register. At any time there will be exactly one molecule of some species S_i corresponding to the current state i , and none of the other species S_j , for $j \neq i$. The molecular count of M_r represents the value of register r . For every $inc(i, r, j)$ instruction we add an inc reaction $S_i \rightarrow S_j + M_r$. For every $dec(i, r, j, k)$ instruction we add two reactions $dec_1: S_i + M_r \rightarrow S_j$ and $dec_2: S_i \rightarrow S_k$. We must ensure that a nonzero register decrements with high probability, which is the only source of error in this simulation. The probability of error per step is $\varepsilon = k_2/(k_1/v + k_2)$ in the worst case that the register holds the value 1, where k_1 is the rate constant for dec_1 and k_2 is the rate constant for dec_2 . To decrease the error, we can increase k_1 , decrease k_2 ,

A		B	
Rxn	Logical function	Rxn	Catalysts
$(inc) \quad C + S_i \rightarrow S_j + M_r + C$	$inc(i, r, j)$	$C_l \rightarrow C_{l-1}$	A^*
$(dec_1) \quad S_i + M_r \rightarrow S_j$	$dec(i, r, j, k)$	$C_{l-1} \rightarrow C_l$	A
$(dec_2) \quad C_1 + S_i \rightarrow S_k + C_l$		\vdots	\vdots
		$C_3 \rightarrow C_2$	A^*
		$C_2 \rightarrow C_3$	A
		$C_2 \rightarrow C_1$	A^*
		$C_1 \rightarrow C_2$	A

Figure 6.1: (A) Bounded RM simulation. Species C ($\#C = 1$) acts as a dummy catalyst to ensure that all reactions are bimolecular, simplifying the analysis of how the simulation scales with the volume. Initial molecular counts are: if \hat{i} is the start state then $\#S_{\hat{i}} = 1$, $\#S_j = 0$ for $j \neq \hat{i}$, and $\#M_r$ is the initial value of register r . (B) Clock module for the RM and CTM simulations. Intuitively, the clock module maintains the average concentration of C_1 at approximately $(\#A^*)^l / (\#A)^{l-1}$. Initial molecular counts are: $\#C_l = 1$, $\#C_1 = \dots = \#C_{l-1} = 0$. For the RM simulation $\#A^* = 1$ and $\#A = \Theta(1/\varepsilon^{1/(l-1)})$. In the RM simulation, A^* acts as a dummy catalyst to ensure that all reactions in the clock module are bimolecular and thus scale equivalently with the volume. This ensures that the error probability is independent of the volume. For the bounded CTM simulation, we use $\#A^* = \Theta((\frac{3^s c_t}{s c_t})^{1/l})$ and $\#A = \Theta((\frac{1}{\varepsilon^{3/2}})^{1/(l-1)})$ (see Section 6.7.3). Because constructions of Section 6.5 will require differing random walk lengths, we allow different values of l .

or decrease the volume v .

Decreasing the volume or changing the rate constants to modify the error rate is problematic. Changing the volume may be impossible (e.g., the volume is that of a cell). Further, a major assumption essential to maintain well-mixedness and justify the given kinetics is that the solution is dilute. The *finite density constraint* implies that the solution volume cannot be arbitrarily small and in fact must be at least proportional to the maximum molecular count attained during computation. Further, since developing new chemistry to perform larger computation is undesirable, improving the error rate of the chemical implementation of an RM without adjusting rate constants is essential.

In every construction to follow, the error probability is determined not by the volume or rate constants, but by the initial molecular count of an “accuracy species” which is easily changed. In fact, we use exclusively bimolecular reactions* and all rate constants are fixed at some arbitrary value k . Using exclusively bimolecular reactions simplifies the analysis of how the speed of the simulation scales with the volume and ensures that the error probability is independent of the volume. Further, working with the added restriction that all rate constants are equal forces us to design robust behavior that does not depend on the precise value of the rate constants.

We modify our first attempt at simulating an RM to allow the arbitrary decrease of error rates by increasing the initial molecular count of the accuracy species A . In the new construction, dec_2 is modified to take a molecule of a new species C_1 as reactant (see Fig 6.1(a)), so that decreasing the effective molecular count of C_1 is essentially equivalent to decreasing the rate constant of the original reaction. While we cannot arbitrarily decrease $\#C_1$ (at the bottom it is either 1 or 0), we can decrease the “average value” of $\#C_1$. Fig 6.1(b) shows a “clock module” that maintains the average value of $\#C_1$ at approximately $(1/\#A)^{l-1}$, where l is the length of the random walk in the clock module (see Lemma 6.7.4 in the Appendix). Thus, to obtain error probability per step ε we use $\#A = \Theta(1/\varepsilon^{1/(l-1)})$ while keeping all rate constants fixed.†

How do we measure the speed of our simulation? We can make the simulation faster by decreasing the volume, finding a physical implementation with larger rate constants, or by increasing the error rate. Of

*All unimolecular reactions can be turned into bimolecular reactions by adding a dummy catalyst.

†The asymptotic notation we use throughout this paper can be understood as follows. We write $f(x, y, \dots) = O(g(x, y, \dots))$ if $\exists c > 0$ such that $f(x, y, \dots) \leq c \cdot g(x, y, \dots)$ for all allowed values of x, y, \dots . The allowed range of the parameters will be given either explicitly, or implicitly (e.g., probabilities must be in the range $[0, 1]$). Similarly, we write $f(x, y, \dots) = \Omega(g(x, y, \dots))$ if $\exists c > 0$ such that $f(x, y, \dots) \geq c \cdot g(x, y, \dots)$ for all allowed values of x, y, \dots . We say $f(x, y, \dots) = \Theta(g(x, y, \dots))$ if both $f(x, y, \dots) = O(g(x, y, \dots))$ and $f(x, y, \dots) = \Omega(g(x, y, \dots))$.

course, there are limits to each of these: the volume may be set (i.e., operating in a cell), the chemistry is what’s available, and, of course, the error cannot be increased too much or else computation is unreliable. As a function of the relevant parameters, the speed of the RM simulation is as given by the following theorem, whose proof is given in Section 6.7.2.

Theorem 6.4.1 (Bounded computation: RM simulation). *For any RM, there is an SCRN such that for any non-zero error probability δ , any input, and any bound on the number of RM steps t , there is an initial amount of the accuracy species A that allows simulation of the RM with cumulative error probability at most δ in expected time $O(\frac{vt^2}{k\delta})$, where v is the volume, and k is the rate constant.*

The major effort of the rest of this section is in speeding up the computation. The first problem is that while we are simulating an RM without much of a slowdown, the RM computation itself is very slow, at least when compared to a Turing machine (TM). For most algorithms t steps of a TM correspond to $\Omega(2^t)$ steps of a RM [27].* Thus, the first question is whether we can simulate a TM instead of the much slower RM? We achieve this in our next construction where we simulate an abstract machine called a clockwise TM (CTM)[28] which is only quadratically slower than a regular TM (Lemma 6.7.9).

Our second question is whether it is possible to speed up computation by increasing the molecular counts of some species. After all, in bulk chemistry reactions can be sped up equivalently by decreasing the volume or increasing the amount of reactants. However, storing information in the exact molecular counts imposes a constraint since increasing the molecular counts to speed up the simulation would affect the information content. This issue is especially important if the volume is outside of our control (e.g., the volume is that of a cell).

A more essential reason for desiring a speed-up with increasing molecular counts is the previously stated finite density constraint that the solution volume should be at least proportional to the maximum molecular count attained in the computation. Since information stored in molecular counts is unary, we require molecular counts exponential in the number of bits stored. Can we ensure that the speed increases with molecular counts enough to compensate for the volume that necessarily must increase as more information is stored?

We will show that the CTM can be simulated in such a manner that increasing the molecular counts of some species does not interfere with the logic of computation and yet yields a speed-up. To get a sense of the speed-up possible, consider the reaction $X + Y \rightarrow Y + \dots$ (i.e., Y is acting catalytically with products that don’t concern us here) with both reactants initially having molecular counts m . This reaction completes (i.e., every molecule of X is used up) in expected time that scales with m as $O(\frac{\log m}{m})$ (Lemma 6.7.5); intuitively, even though more X must be converted for larger m , this is an exponential decay process of X occurring at rate $O(\#Y) = O(m)$. Thus by increasing m we can speed it up almost linearly. By ensuring that all reactions in a step of the simulation are of this form, or complete just as quickly, we guarantee that by increasing m we can make the computation proceed faster. The almost linear speed-up also adequately compensates for the volume increasing due to the finite density constraint.

For the purposes of this paper, a TM is a finite state machine augmented with a two-way infinite tape, with a single head pointing at the current bit on the tape. A TM instruction combines reading, writing, changing the state, and moving the head. Specifically, the instruction $op(i, j, k, z_j, z_k, D)$ specifies that if starting in state i , first read the current bit and change to either state j if it is 0 or state k if it is 1, overwrite the current bit with z_j or z_k respectively, and finally move the head left or right along the tape as indicated by the direction D . It is well known that a TM can be simulated by an “enhanced” RM in linear time if the operations of multiplication by a constant and division by a constant with remainder can also be done as one-step operations. To do this, the content of the TM tape is represented in the binary expansion of two register values (one for the bits to the left of the head and one for the bits to the right, with low-order bits representing tape symbols near the TM head, and high-order bits representing symbols far from the head). Simulating the motion of the head involves division and multiplication by the number base (2 for a binary TM) of the respective registers because these operations correspond to shifting the bits right or left. In a SCRN, multiplication by 2 can be done by a reaction akin to $M \rightarrow 2M'$ catalyzed by a species of comparable number of molecules, which has the fast kinetics of the $X + Y \rightarrow Y + \dots$ reaction above. However, performing division quickly enough

*By the (extended) Church-Turing thesis, a TM, unlike a RM, is the best we can do, if we care only about super-polynomial distinctions in computing speed.

seems difficult in a SCRN.* To avoid division, we use a variant of a TM defined as follows. A CTM is a TM-like automaton with a finite circular tape and instructions of the form $op(i, j, k, z_j, z_k)$. The instruction specifies behavior like a TM, except that the head always moves clockwise along the tape. Any TM with a two-way infinite tape using at most s_{tm} space and t_{tm} time can easily be converted to a clockwise TM using no more than $s_{ct} = 2s_{tm}$ space and $t_{ct} = O(t_{tm}s_{tm})$ time (Lemma 6.7.9). The instruction $op(i, j, k, z_j, z_k)$ corresponds to: if starting in state i , first read the most significant digit and change to either state j if it is 0 or state k if it is 1, erase the most significant digit, shift all digits left via multiplying by the number base, and finally write a new least significant digit with the value z_j if the most significant digit was 0 or z_k if it was 1. Thus, instead of dividing to shift bits right, the circular tape allows arbitrary head movement using only the left bit shift operation (which corresponds to multiplication).

The reactions simulating a CTM are shown in Fig. 6.2. Tape contents are encoded in the base-3 digit expansion of $\#M$ using digit 1 to represent binary 0 and digit 2 to represent binary 1. This base-3 encoding ensures that reading the most significant bit is fast enough (see below). To read the most significant digit of $\#M$, it is compared with a known threshold quantity $\#T$ by the reaction $M + T \rightarrow \dots$ (such that either T or M will be in sufficient excess, see below). We subdivide the CTM steps into microsteps for the purposes of our construction; there are four microsteps for a CTM step. The current state and microstate is indicated by which of the state species $\{S_{i,z}\}$ is present, with i indicating the state CTM finite control and $z \in \{1, 2, 3, 4\}$ indicating which of the four corresponding microstates we are in. The division into microsteps is necessary to prevent potentially conflicting reactions from occurring simultaneously as they are catalyzed by different state species and thus can occur only in different microsteps. Conflicting reactions are separated by at least two microsteps, since during the transition between two microsteps there is a time when both state species are present. A self-catalysis chain reaction is used to move from one microstep to the next. The transition is initiated by a reaction of a state species with a clock molecule C_1 to form the state species corresponding to the next microstep.

Now with $m = 3^{s_{ct}-1}$, Lemmas 6.7.5–6.7.7 guarantee that all reactions in a microstep (excluding state transition initiation reactions) complete in expected time $O(\frac{v \log m}{km}) = O(\frac{vs_{ct}}{k3^{s_{ct}}})$. Specifically, Lemma 6.7.5 ensures that the memory operation reactions having a state species as a catalyst complete in the required time. Lemma 6.7.7 does the same for the self-catalytic state transition reactions. Finally, ensuring that either M or T is in excess of the other by $\Theta(m)$ allows us to use Lemma 6.7.6 to prove that the reading of the most significant bit occurs quickly enough. The separation of $\#M$ or $\#T$ is established by using values of $\#M$ expressed in base 3 using just the digits 1 and 2. Then the threshold value $\#T$ as shown in Fig. 6.2 is $\Theta(3^{s_{ct}})$ larger than the largest possible s_{ct} -digit value of $\#M$ starting with 1 (base-3) and $\Theta(3^{s_{ct}})$ smaller than the smallest possible s_{ct} -digit value of $\#M$ starting with 2 (base-3), implying that either T or M will be in sufficient excess.

The only source of error is if not all reactions in a microstep finish before a state transition initiation reaction occurs. This error is controlled in an analogous manner to the RM simulation: state transition initiation reactions work on the same principle as the delayed dec_2 reaction of the RM simulation. We adjust $\#A$ so that all reactions in a microstep have a chance to finish before the system transitions to the next microstep (see Section 6.7.3).

Since as a function of s_{ct} , the reactions constituting a microstep in the CTM simulation finish in expected time $O(\frac{vs_{ct}}{k3^{s_{ct}}})$, by increasing s_{ct} via padding of the CTM tape with extra bits we can decrease exponentially the amount of time we need to allocate for each microstep. This exponential speed-up is only slightly dampened by the increase in the number of CTM steps corresponding to a single step of the TM (making the worst case assumption that the padded bits must be traversed on every step of the TM, Lemma 6.7.9).

In total we obtain the following result (see Section 6.7.3). It shows that we can simulate a TM with only a polynomial slowdown, and that computation can be sped up by increasing the molecular count of some species through a “padding parameter” Δ .

Theorem 6.4.2 (Bounded computation: TM simulation). *For any TM, there is an SCRN such that for any non-zero error probability δ , any amount of padding Δ , any input, any bound on the number of TM steps t_{tm} , and any bound on TM space usage s_{tm} , there is an initial amount of the accuracy species A that allows*

*For example, the naive approach of dividing $\#M$ by 2 by doing $M + M \rightarrow M'$ takes $\Theta(1)$ time (independent of $\#M$) as a function of the initial amount of $\#M$. Note that the expected time for the last two remaining M s to react is a constant. Thus, if this were a step of our TM simulation we would not attain the desired speed-up with increasing molecular count.

	Rxn	Catalysts	Logical function
State transitions	$C_1 \rightarrow C_1 + I$	A^*	State transition initiation reaction
	$I + S_{i,4} \rightarrow S_{i,1}$	\emptyset	State $(i, 4) \rightarrow (i, 1)$ transition
	$S_{i,4} \rightarrow S_{i,1}$	$S_{i,1}$	
	$I + S_{i,1} \rightarrow S_{i,2}$	\emptyset	State $(i, 1) \rightarrow (i, 2)$ transition
	$S_{i,1} \rightarrow S_{i,2}$	$S_{i,2}$	
	$I + S_{i,2} \rightarrow S_{i,3}$	\emptyset	State $(i, 2) \rightarrow (i, 3)$ transition
	$S_{i,2} \rightarrow S_{i,3}$	$S_{i,3}$	
	$I \rightarrow I_T$	T	State $(i, 3) \rightarrow (j, 4)$ transition if high order bit is 0
	$I_T + S_{i,3} \rightarrow S_{j,4}$	\emptyset	
	$S_{i,3} \rightarrow S_{j,4}$	$S_{j,4}$	
$I \rightarrow I_M$	M	State $(i, 3) \rightarrow (k, 4)$ transition if high order bit is 1	
$I_M + S_{i,3} \rightarrow S_{k,4}$	\emptyset		
$S_{i,3} \rightarrow S_{k,4}$	$S_{k,4}$		
Memory operations	$M^\dagger \rightarrow M$	$S_{i,2}$	Restore memory
	$T^\dagger \rightarrow T$	$S_{i,2}$	Compare memory to threshold
	$T + M \rightarrow T^* + M^*$	\emptyset	
	$D^* \rightarrow D$	$S_{i,2}$	Restore D and B
	$B^* \rightarrow B$	$S_{i,2}$	
	$T^* \rightarrow T^\dagger$	$S_{j,4}$ or $S_{k,4}$	Restore memory and threshold
	$M^* \rightarrow M$	$S_{j,4}$ or $S_{k,4}$	
	$D \rightarrow K_1 + D^*$	$S_{j,4}$	Zero out high order bit
	$D \rightarrow K_2 + D^*$	$S_{k,4}$	
	$K_2 + M \rightarrow K_1$	\emptyset	
$K_1 + M \rightarrow \emptyset$	\emptyset		
$B \rightarrow (z_j + 1)M + B^*$	$S_{j,4}$	Write new low order bit $(z_j, z_k \in \{0, 1\})$	
$B \rightarrow (z_k + 1)M + B^*$	$S_{k,4}$		
$M \rightarrow 3M^\dagger$	$S_{j,4}$ or $S_{k,4}$	Shift digits left	

B	Initial molecular counts	Base 3 representation
	$\#M = (b_1 + 1) \cdot 3^{s-1} + (b_2 + 1) \cdot 3^{s-2} + \dots$ $\dots + (b_{s-1} + 1) \cdot 3 + (b_s + 1)$	$\begin{pmatrix} 1 \\ + \\ \dots \\ + \\ \dots \\ + \end{pmatrix} \begin{pmatrix} 1 \\ + \\ \dots \\ + \\ \dots \\ + \end{pmatrix} \begin{pmatrix} 1 \\ + \\ \dots \\ + \\ \dots \\ + \end{pmatrix} \begin{pmatrix} 1 \\ + \\ \dots \\ + \\ \dots \\ + \end{pmatrix} \begin{pmatrix} 1 \\ + \\ \dots \\ + \\ \dots \\ + \end{pmatrix} \begin{pmatrix} 1 \\ + \\ \dots \\ + \\ \dots \\ + \end{pmatrix}$
	$\#T = 2 \cdot 3^{s-1} + 2 \cdot 3^{s-3}$	$\begin{pmatrix} 2 & 0 & 2 & 0 & 0 & \dots & 0 \end{pmatrix}$
	$\#D = \#S_{i,1} = 3^{s-1}$	$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \end{pmatrix}$
	$\#B = 1$	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & \dots & 1 \end{pmatrix}$

Figure 6.2: Bounded CTM simulation: reactions and initial molecular counts. (A) Reactions for $op(i, j, k, z_j, z_k)$. The clock module is the same as for the RM simulation (Fig. 6.1(B)). Here \emptyset indicates “nothing.” (B) Letting $s = s_{ct}$, initial molecular counts for binary input $b_1 b_2 \dots b_s$. Input is padded with zeros to be exactly s bits long. Here \hat{i} is the start state of the CTM. All species not shown start at 0.

simulation of the TM with cumulative error probability at most δ in expected time $O\left(\frac{v(s_{tm}+\Delta)^{7/2}t_{tm}^{5/2}}{k3^{(2s_{tm}+\Delta)}\delta^{3/2}}\right)$, where v is the volume, and k is the rate constant.

Under realistic conditions relating v , s_{tm} , and t_{tm} , this theorem implies that the SCRN simulates the TM in polynomial time, specifically $O(t_{tm}^6)$. The finite density constraint introduced earlier requires that the solution volume be proportional to the maximum molecular count attained in the course of the computation. This constraint limits the speed of the simulation: there is a minimum volume to implement a particular computation, and if the volume is larger than necessary, the finite density constraint bounds Δ . In most cases, the total molecular count will be dominated by $3^{2s_{tm}+\Delta}$ (see Section 6.7.3). Thus the maximum allowed padding satisfies $3^{2s_{tm}+\Delta} = \Theta(v)$, yielding total expected computation time $O\left(\frac{(\log v)^{7/2}t_{tm}^{5/2}}{k\delta^{3/2}}\right)$. This implies that although Δ cannot be used to speed up computation arbitrarily, it can be used to minimize the effect of having a volume much larger than necessary since increasing the volume decreases the speed of computation poly-logarithmically only. Alternatively, if we can decrease the volume as long as the maximum density is bounded by some constant, then the best speed is obtained with zero padding and the smallest v possible: $v = \Theta(3^{2s_{tm}})$. Then the total computation time is $O\left(\frac{s_{tm}^{7/2}t_{tm}^{5/2}}{k\delta^{3/2}}\right)$. Since we can always ensure $s_{tm} \leq t_{tm}$, we experience at most a polynomial (6th-order) slowdown overall compared with a regular error-free TM.

6.5 Unbounded Algorithms

The above simulations are not Turing universal because they incur a fixed probability of error per step of the computation. Since the probability of correct computation decreases exponentially with the number of steps, only finite computation may be performed reliably. Additionally the TM simulation has the property that the tape size must be specified a priori. We now prove that a fixed SCRN can be Turing universal with an arbitrarily small probability of error over an unlimited number of steps. In the course of a computation that is not a priori bounded, in addition to stirring faster and injecting energy and raw materials, the volume needs to grow at least linearly with the total molecular count to maintain finite density. Therefore, in this section our model is that the volume dynamically changes linearly with the total molecular count as the system evolves over time. We desire that the total error probability over arbitrarily long computation does not exceed δ and can be set by increasing the initial molecular counts of the accuracy species A .

We now sketch how to modify our constructions to allow Turing universal computation. Consider the RM simulation first. We can achieve a bounded total probability of error over an unbounded number of steps by sufficiently decreasing the probability of error in each subsequent error-prone step. Only dec steps when the register is non-zero are error-prone. Further, if dec_2 occurs then either the register value was zero and no error was possible, or an error has just occurred and there is no need to decrease the error further. Therefore it is sufficient to decrease the probability of error after each dec_1 step by producing A as a product of dec_1 . If the clock Markov chain length is $l = 3$, then adding a single molecule of A as a product of every dec_1 reaction is enough: the total probability of error obtained via Lemma 6.7.4 is $O(\sum_{\#A=i_0}^{\infty} 1/\#A^2)$; since this sum converges, the error probability over all time can be bounded by any desired $\delta > 0$ by making the initial number of A s, i_0 , sufficiently large. Using $l = 3$ is best because $l > 3$ unnecessarily slows down the simulation. The total expected computation time is then $O(t(1/\delta + t)^2(1/\delta + t + s_0)/k)$, where s_0 is the sum of the initial register counts (see Section 6.7.4).

A similar approach can be taken with respect to the TM simulation. The added difficulty is that the tape size must no longer be fixed, but must grow as needed. This can be achieved if the SCRN triples the molecular count of the state species, M , T , D , and P whenever the tape needs to increase by an extra bit. However, simply increasing $\#A$ by 1 per microstep without changing $\#A^*$ as in the RM construction does not work since the volume may triple in a CTM step. Then the clock would experience an exponentially increasing expected time. To solve this problem, in Section 6.7.5 we show that if the SCRN triples the amount of A and A^* whenever extending the tape and increases $\#A$ by an appropriate amount, $\Theta(3^{set})$, on every step then it achieves a bounded error probability over all time and yields the running time claimed in Theorem 6.5.1 below. The clock Markov chain of length $l = 5$ is used. All the extra operations can be implemented by reactions similar to the types of reactions already implementing the CTM simulation (Fig. 6.2). For example, tripling A can be done by reactions akin to $A \rightarrow A^\dagger$ and $A^\dagger \rightarrow 3A$ catalyzed by different state species in two

non-consecutive microsteps.*

Theorem 6.5.1 (Turing universal computation). *For any TM, there is an SCRN such that for any non-zero error probability δ , and any bound s_{tm0} on the size of the input, there is an initial amount of the accuracy species A that allows simulation of the TM on inputs of size at most s_{tm0} with cumulative error probability at most δ over an unbounded number of steps and allowing unbounded space usage. Moreover, in the model where the volume grows dynamically in proportion with the total molecular count, t_{tm} steps of the TM complete in expected time (conditional on the computation being correct) of $O((1/\delta + s_{tm0} + t_{tm}s_{tm})^5 t_{tm}s_{tm}/k)$ where s_{tm} is the space used by the TM, and k is the rate constant.*

For $s_{tm} \approx t_{tm}$ this gives a polynomial time (12th-order) simulation of TMs. This slowdown relative to Theorem 6.4.2 is due to our method of slowing down the clock to reduce errors.

Can SCRN achieve Turing universal computation without error? Can we ask for a guarantee that the system will eventually output a correct answer with probability 1?[†] Some simple computations are indeed possible with this strong guarantee, but it turns out that for general computations this is impossible. Intuitively, when storing information in molecular counts, the system can never be sure it has detected all the molecules present, and thus must decide to produce an output at some point without being certain. Formally, a theorem due to Karp and Miller [18] when adapted to the SCRN context (see Section 6.7.6) rules out the possibility of error-free Turing universal computation altogether if the state of the TM head can be determined by the presence or absence (or threshold quantities) of certain species (i.e., state species in our constructions). Here recall that in computer science a question is called decidable if there is an algorithm (equivalently TM) that solves it in all cases. (Recall a state of a SCRN is a vector of molecular counts of each of the species. Below operator \geq indicates element-wise comparison.)

Theorem 6.5.2. *For any SCRN, given two states \mathbf{x} and \mathbf{y} , the question of whether any state $\mathbf{y}' \geq \mathbf{y}$ is reachable from \mathbf{x} is decidable.*

How does this theorem imply that error-free Turing universal computation is impossible? Since all the constructions in this paper rely on probabilities we need to rule out more clever constructions. First recall that a question is undecidable if one can prove that there can be no algorithm that solves it correctly in all cases; the classic undecidable problem is the Halting problem: determine whether or not a given TM will eventually halt [33]. Now suppose by way of contradiction that someone claims to have an errorless way of simulating any TM in a SCRN. Say it is claimed that if the TM halts then the state species corresponding to the halt state is produced with non-zero probability (this is weaker than requiring probability 1), while if the TM never halts then the halt state species cannot be produced. Now note that by asking whether a state with a molecule of the halting species is reachable from the initial state, we can determine whether the TM halts: if such a state is reachable then there must be a finite sequence of reactions leading to it, implying that the probability of producing a halt state species is greater than 0; otherwise, if such a state is not reachable, the halt state species can never be produced. This is equivalent to asking whether we can reach any $\mathbf{y}' \geq \mathbf{y}$ from the initial state of the SCRN, where \mathbf{y} is the all zero vector with a one in the position of the halting species — a question that we know is always computable, thanks to Karp and Miller. Thus if an errorless way of simulating TMs existed, we would violate the undecidability of the halting problem.

Finally note that our Turing-universality results imply that the their long-term behavior of SCRN is unknowable in a probabilistic sense. Specifically, our results imply that the question of whether a given SCRN, starting with a given initial state \mathbf{x} , produces a molecule of a given species with high or low probability is in general undecidable. This can be shown using a similar argument: if the question were decidable the halting problem could be solved by encoding a TM using our construction, and asking whether the SCRN eventually produces a molecule of the halting state species.

A slight modification of the clock module is necessary to maintain the desired behavior. Because of the need of intermediate species (e.g., A^\dagger) for tripling $\#A$ and $\#A^$, the clock reactions need to be catalyzed by the appropriate intermediate species in addition to A and A^* .

[†]Since a reaction might simply not be chosen for an arbitrarily long time (although the odds of this happening decrease exponentially), we can't insist on a zero probability of error at any fixed time.

6.6 Discussion

We show that computation on molecular counts in the SCRN model of stochastic chemical kinetics can be fast, in the sense of being only polynomially slower than a TM, and accurate, in the sense that the cumulative error probability can be made arbitrarily small. Since the simulated TM can be universal [33], a single set of species and chemical reactions can perform any computation that can be performed on any computer. The error probability can be manipulated by changing the molecular count of an accuracy species, rather than changing the underlying chemistry. Further, we show that computation that is not a priori bounded in terms of time and space usage can be performed assuming that the volume of the solution expands to accommodate the increase in the total molecular count. In other words SCRNs are Turing universal.

The Turing-universality of SCRNs implies that the question of whether given a start state the system is *likely* to produce a molecule of a given species is in general undecidable. This is contrasted with questions of possibility rather than probability: whether a certain molecule *could* be produced is always decidable.

Our results may imply certain bounds on the speed of stochastic simulation algorithms (such as variants of τ -leaping [16]), suggesting an area of further study. The intuition is as follows: it is well known by the time hierarchy theorem [33] that certain TMs cannot be effectively sped up (it is impossible to build a TM that has the same input/output relationship but computes much faster). This is believed to be true even allowing some probability of error [4]. Since a TM can be encoded in an SCRN, if the behavior of the SCRN could be simulated very quickly, then the behavior of the TM would also be determined quickly, which would raise a contradiction.

Our results were optimized for clarity rather than performance. In certain cases our running time bounds can probably be significantly improved (e.g., in a number of places we bound additive terms $O(x+y)$, where $x \geq 1$ and $y \geq 1$, by multiplicative terms $O(xy)$). Also the roles of a number of species can be performed by a single species (e.g., A^* and C in the RM simulation).

A number of previous works have attempted to achieve Turing-universality with chemical kinetics. However, most proposed schemes require increasing the variety of molecular species (rather than only increasing molecular counts) to perform larger computation (e.g., [24] which shows finite circuit computation and not Turing universal computation despite its title). Liekens and Fernando [22] have considered computation in stochastic chemistry in which computation is performed on molecular counts. Specifically, they discuss how SCRNs can simulate RMs. However, they rely on the manipulation of rate constants to attain the desired error probability per step. Further, they do not achieve Turing universal computation, as the prior knowledge of the length of the computation is required to set the rate constants appropriately to obtain a desired total error probability. While writing this manuscript, the work of Angluin et al. [2] in distributed computing and multi-agent systems came to our attention. Based on the formal relation between their field and our field, one concludes that their results imply that stochastic chemical reaction networks can simulate a TM with a polynomial slowdown (a result akin to our Theorem 6.4.2). Compared to our result, their method allows attaining a better polynomial (lower degree), and much better dependence on the allowed error probability (e.g., to decrease the error by a factor of 10, we have to slow down the system by a factor of $10^{3/2}$, while an implementation based on their results only has to slow down by a factor polynomial in $\log 10$). However, because we focus on molecular interactions rather than the theory of distributed computing, and measure physical time for reaction kinetics rather than just the number of interactions, our results take into account the solution volume and the consequences of the finite density constraint (Section 6.4). Further, while they consider only finite algorithms, we demonstrate Turing-universality by discussing a way of simulating algorithms unbounded in time and space use (Section 6.5). Finally, our construction is simpler in the sense that it requires far fewer reactions. The relative simplicity of our system makes implementing Turing universal chemical reactions a plausible and important goal for synthetic biology.

6.7 Appendix

6.7.1 Clock Analysis

The following three lemmas refer to the Markov chain in Fig. 6.3. We use $p_i(t)$ to indicate the probability of being in state i at time t . CDF stands for cumulative distribution function.

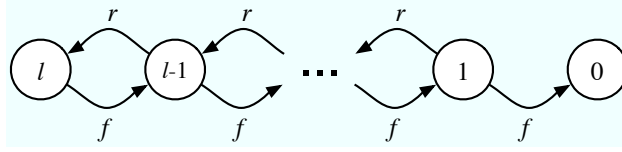


Figure 6.3: Continuous-time Markov chain for Lemmas 6.7.1–6.7.3. States $i = 1, \dots, l$ indicate the identity of the currently present clock species C_1, \dots, C_l . Transition to state 0 represents reaction dec_2 for the RM simulation or the state transition initiation reaction of the CTM simulation.

Lemma 6.7.1. *Suppose the process starts in state l . Then $\forall t, p_1(t) \leq (1 - p_0(t))\mu$ where $\mu = 1/(1 + \frac{r}{f} + (\frac{r}{f})^2 + \dots + (\frac{r}{f})^{l-1})$.*

Proof. Consider the Markov chain restricted to states $1, \dots, l$. We can prove that the invariance $p_{i+1}(t)/p_i(t) \geq r/f$ (for $i = 1, \dots, l-1$) is maintained at all times through the following argument. Letting $\phi_i(t) = p_{i+1}(t)/p_i(t)$, we can show $d\phi_i(t)/dt \geq 0$ when $\phi_i(t) = r/f$ and $\forall i', \phi_{i'}(t) \geq r/f$, which implies that for no i can $\phi_i(t)$ fall below r/f if it starts above. This is done by showing that $dp_i(t)/dt = p_{i+1}(t)f + p_{i-1}(t)r - (r+f)p_i(t) \leq 0$ since $\phi_i(t) = r/f$ and $\phi_{i-1}(t) \geq r/f$, and $dp_{i+1}(t)/dt = p_{i+2}(t)f + p_i(t)r - (r+f)p_{i+1}(t) \geq 0$ since $\phi_i(t) = r/f$ and $\phi_{i+1}(t) \geq r/f$ (the p_{i-1} or the p_{i+2} terms are zero for the boundary cases).

Now $p_i(t) = \phi_{i-1}(t)\phi_{i-2}(t) \cdots \phi_1(t)p_1(t)$. Thus $\sum_i p_i(t) = 1$ implies $p_1(t) = 1/(1 + \phi_1 + \phi_2\phi_1 + \dots + \phi_{l-1}\phi_{l-2} \cdots \phi_1) \leq 1/(1 + \frac{r}{f} + (\frac{r}{f})^2 + \dots + (\frac{r}{f})^{l-1})$. This is a bound on the probability of being in state 1 given that we haven't reached state 0 in the full chain of Fig. 6.3. Thus multiplying by $1 - p_0(t)$ gives us the desired result. \square

Lemma 6.7.2. *Suppose for some μ we have $\forall t, p_1(t) \leq (1 - p_0(t))\mu$. Let T be a random variable describing the time until absorption at state 0. Then $\Pr[T < t] \leq 1 - e^{-\lambda t}$ for $\lambda = f\mu$ (i.e., our CDF is bounded by the CDF for an exponential random variable with rate $\lambda = f\mu$).*

Proof. The result follows from the fact that $dp_0(t)/dt = p_1(t)f \leq (1 - p_0(t))\mu f$. \square

Lemma 6.7.3. *Starting at state l , the expected time to absorb at state 0 is $O((\frac{r}{f})^{l-1}/f)$ assuming sufficiently large r/f .*

Proof. The expected number of transitions to reach state 0 starting in state i is $d_i = \frac{2pq((q/p)^l - (q/p)^{l-i})}{(1-2p)^2} - \frac{i}{q-p}$, where $p = \frac{f}{f+r}$ is the probability of transitioning to a state to the left and $q = 1 - p$ is the probability of transitioning to the state to the right. This expression is obtained by solving the recurrence relation $d_i = pd_{i-1} + qd_{i+1} + 1$ ($0 > i > l$) with boundary conditions $d_0 = 0, d_l = d_{l-1} + 1$. Thus $d_l < \frac{2pq(q/p)^l}{(1-2p)^2} = \frac{2(r/f)^{l+1}}{(r/f-1)^2}$. This implies that for r/f larger than some constant, $d_l = O((\frac{r}{f})^{l-1})$. Since the expected duration of any transition is no more than $1/f$, the desired bound is obtained. \square

By the above lemmas, the time for the clock to “tick” can be effectively thought of as an exponential random variable with rate $\lambda = f/(1 + \frac{r}{f} + (\frac{r}{f})^2 + \dots + (\frac{r}{f})^{l-1}) = \Theta(\frac{f}{(r/f)^{l-1}})$. Lemma 6.7.2 shows that the CDF of the tick is bounded by the CDF of this exponential random variable. Further, Lemma 6.7.3 shows that the expected time for the tick is bounded by (the order of) expected time of this exponential random variable. Note that Lemma 6.7.2 is true no matter how long the clock has already been running (a “memoryless” property). For our clock construction (Fig. 6.1(b)), we set λ by changing $\#A$ and $\#A^*$ which define the forward and reverse rates f and r . Specifically, we have $\lambda = \Theta(\frac{k\#A^*}{v\#A^{l-1}})$.

6.7.2 Time/Space-Bounded RM Simulation

Lemma 6.7.4. *For the finite RM simulation, the probability of error per step is $O((1/\#A)^{l-1})$. Further, the expected time per step is bounded by $O((\#A)^{l-1}v/k)$.*

Proof. Consider the point in time when the RM simulation enters a state in which it should decrement a non-empty register. If the time until dec_2 occurs were an exponential random variable with rate λ then the probability of error per step would be bounded by $\lambda/(k/v + \lambda)$. (We are making the worst case assumption that there is exactly one register molecule; otherwise, the error is even smaller.) The time until dec_2 is not exponentially distributed, but by Section 6.7.1, it can be bounded by an exponential random variable with rate $\lambda = O(\frac{k}{v\#A^{l-1}})$ ($\#A^* = 1$ for the RM construction). Note that the clock may have been running for a while since the last dec operation (while the RM performs inc operations for example); however, this amount of time is irrelevant by the memoryless property established in Section 6.7.1. Thus the probability of error per step is bounded by $\lambda/(k/v + \lambda) = O((1/\#A)^{l-1})$. The expected time per RM step is bounded by the expected time for dec_2 which is $O((\#A)^{l-1}v/k)$ by Section 6.7.1. \square

The above lemma implies that we can use $\#A = \Theta((t/\delta)^{1/(l-1)})$ resulting in the expected time for the whole computation of $O(\frac{vt^2}{k\delta})$ and the total probability of error being bounded by δ .

6.7.3 Time/Space-Bounded CTM Simulation

In the following lemmas, we say a reaction *completely finishes* when it happens enough times that one of the reactants is used up.

Lemma 6.7.5. *Starting with $\Theta(m)$ molecules of X and $\Theta(m)$ molecules of Y , the expected time for the reaction $X + Y \rightarrow Y$ to completely finish is $O(\frac{v}{km} \log m)$. The variance of the completion time is $O((\frac{v}{km})^2)$.*

Proof. When there are q molecules of X remaining, the waiting time until next reaction is an exponential random variable with rate $\Theta(kqm/v)$ and therefore mean $O(\frac{v}{kqm})$. Each waiting time is independent. Thus the total expected time is $\sum_{q=1}^{\Theta(m)} O(\frac{v}{kqm}) = O(\frac{v}{km} \log m)$.^{*} The variance of each waiting time is $O((\frac{v}{kqm})^2)$. Thus the total variance is $\sum_{q=1}^{\Theta(m)} O((\frac{v}{kqm})^2) = O((\frac{v}{km})^2)$. \square

Lemma 6.7.6. *Starting with $\Theta(m)$ molecules of X and $\Theta(m)$ molecules of Y such that $\Delta = \#Y - \#X = \Omega(m)$ the expected time for the reaction $X + Y \rightarrow \emptyset$ to completely finish is $O(\frac{v}{km} \log m)$. The variance of the completion time is $O((\frac{v}{km})^2)$.*

Proof. This case can be proven by reducing to Lemma 6.7.5 with initial amounts $\#Y' = \Delta$ and $\#X' = \#X$. \square

Lemma 6.7.7. *Starting with $\Theta(m)$ molecules of X and 1 molecule of Y , the expected time for the reaction $X + Y \rightarrow 2Y$ to completely finish is $O(\frac{v}{km} \log m)$. The variance of the completion time is $O((\frac{v}{km})^2)$.*

Proof. Consider splitting the process into two halves, with the first part bringing the amount of X to half its initial value and the second part using up the remainder. The time-reverse of the first part, as well as the second part, can both be bounded by processes covered by Lemma 6.7.5. (Assume that $\#X$ is fixed at its minimal value for part one, and assume $\#Y$ is fixed at its minimal value for part two. The variance can only decrease.) \square

Lemma 6.7.8. *Some $\lambda = \Theta(\frac{k\varepsilon^{3/2}3^{s_{ct}}}{vs_{ct}})$ attains error at most ε per microstep of the CTM simulation.*

Proof. Using the above lemmas with $m = 3^{s_{ct}-1}$, by Chebyshev's inequality,[†] with probability at least $1 - \varepsilon/2$ all reactions finish before some time $t_f = \Theta(\frac{v}{km}(\log(m) + 1/\sqrt{\varepsilon})) = O(\frac{v \log m}{km\varepsilon^{1/2}})$. Now we set λ such that the probability that the clock ticks before time t_f is smaller than $\varepsilon/2$ (for a total probability of error ε). Since the time until the clock ticks is bounded by the CDF of an exponential random variable with rate λ (Sec 6.7.1), it is enough that $\lambda < \frac{\varepsilon}{2t_f}$ and so we can choose some $\lambda = \Theta(\frac{\varepsilon^{3/2}km}{v \log m})$. \square

^{*}As $m \rightarrow \infty$, the difference between $\sum_{q=1}^m (1/q)$ and $\log m$ approaches the Euler-Mascheroni constant.

[†]Chebyshev's inequality states that for a random variable X with expected value μ and finite variance σ^2 , for any $d > 0$, $\Pr[|X - \mu| \geq d\sigma] \leq 1/d^2$.

Lemma 6.7.9. Any TM with a two-way infinite tape using at most s_{tm} space and t_{tm} time can be converted to a CTM using $s_{ct} = 2s_{tm}$ space and $t_{ct} = O(t_{tm}s_{tm})$ time. If Δ extra bits of padding on the CTM tape is used, then $t_{ct} = O(t_{tm}(s_{tm} + \Delta))$ time is required.

Proof. (sketch, see [28]) Two bits of the CTM are used to represent a bit of the TM tape. The extra bit is used to store a TM head position marker. To move in the direction corresponding to moving the CTM head clockwise (the easy direction) is trivial. To move in the opposite direction, we use the temporary marker to record the current head position and then move each tape symbol clockwise by one position. Thus, a single TM operation in the worst case corresponds to $O(s)$ CTM operations. \square

In order to simulate t_{tm} steps of a TM that uses s_{tm} bits of space on a CTM using Δ bits of padding requires $t_{ct} = O(t_{tm}(s_{tm} + \Delta))$ CTM steps and a circular tape of size $s_{ct} = 2s_{tm} + \Delta$ (Lemma 6.7.9). Recall that in our CTM simulation, there are four microsteps corresponding to a single CTM operation, which is asymptotically still $O(t_{ct})$. Thus, in order for the total error to be at most δ , we need the error per CTM microstep to be $\varepsilon = O(\frac{\delta}{t_{tm}(s_{tm} + \Delta)})$. Setting the parameters of the clock module ($\#A, \#A^*$) to attain the

largest λ satisfying Lemma 6.7.8, the expected time per microstep is $O(\frac{vs_{ct}}{k3^{s_{ct}}\varepsilon^{3/2}}) = O(\frac{v(s_{tm} + \Delta)^{5/2}t_{tm}^{3/2}}{k3^{2s_{tm} + \Delta}\delta^{3/2}})$. This can be done, for example, by setting $\#A^{*l} = \Theta(\frac{3^{s_{ct}}}{s_{ct}})$ and $\#A^{l-1} = \Theta(\frac{1}{\varepsilon^{3/2}})$. Since there are total $O(t_{tm}(s_{tm} + \Delta))$ CTM microsteps, the total expected time is $O(\frac{v(s_{tm} + \Delta)^{7/2}t_{tm}^{5/2}}{k3^{(2s_{tm} + \Delta)}\delta^{3/2}})$.

How large is the total molecular count? If we keep δ constant while increasing the complexity of the computation being performed, and setting $\#A^*$ and $\#A$ as suggested above, we have that the total molecular count is $\Theta(m + \#A)$ where $m = 3^{2s_{tm} + \Delta}$. Now m increases at least exponentially with $s_{tm} + \Delta$, while $\#A$ increases at most polynomially. Further, m increases at least quadratically with t_{tm} (for any reasonable algorithm $2^{s_{tm}} \geq t_{tm}$) while $\#A$ increases at most as a polynomial of degree $(3/2)\frac{1}{l-1} < 2$. Thus m will dominate.

6.7.4 Unbounded RM Simulation

After i dec_2 steps, we have $\#A = i_0 + i$ where i_0 is the initial number of As. The error probability for the next step is $O(1/\#A^2) = O(1/(i_0 + i)^2)$ by Lemma 6.7.4 when $l = 3$. The total probability of error over an unbounded number of steps is $O(\sum_{i=0}^{\infty} 1/(i_0 + i)^2)$. To make sure this is smaller than δ we start out with $i_0 = \Theta(1/\delta)$ molecules of A^* .

Now what is the total expected time for t steps? By Lemma 6.7.4 the expected time for the next step after i dec_2 steps is $O(\#A^2 v/k) = O((i_0 + i)^2 v/k)$. Since each step at most increases the total molecular count by 1, after t total steps v is not larger than $O(i_0 + t + s_0)$, where s_0 is the sum of the initial values of all the registers. Thus the expected time for the t th step is bounded by $O((i_0 + i)^2(i_0 + t + s_0)/k) = O((1/\delta + t)^2(1/\delta + t + s_0)/k)$ and so the expected total time for t steps is $O(t(1/\delta + t)^2(1/\delta + t + s_0)/k)$.

6.7.5 Unbounded CTM Simulation

We want to follow a similar strategy as in the RM simulation (Section 6.7.4) and want the error probability on the i th CTM step to be bounded by $\varepsilon = 1/(\Theta(1/\delta) + i)^2$ such that the total error probability after arbitrarily many steps is bounded by δ . By Lemma 6.7.8, we can attain per-step error probability (taking the union bound over the 4 microsteps in a step) bounded by this ε when we choose a small enough $\lambda = \Theta(\frac{k\varepsilon^{3/2}3^{s_{ct}}}{vs_{ct}}) = \Theta(\frac{k3^{s_{ct}}}{v(1/\delta + i)^3 s_{ct}})$, where s_{ct} is the current CTM tape size. Recall that λ is set by $\#A$ and $\#A^*$ such that $\lambda = \Theta(\frac{k\#A^*l}{v\#A^{l-1}})$ (Section 6.7.1). It is not hard to see that we can achieve the desired λ using clock Markov chain length $l = 5$, and appropriate $\#A = \Theta((i_0 + i)3^{s_{ct}})$ and $\#A^* = \Theta(3^{s_{ct}})$, for appropriate $i_0 = \Theta(1/\delta + s_{ct0})$, where s_{ct0} is the initial size of the tape. These values of $\#A$ and $\#A^*$ can be attained if the SCRNs triples the amount of A and A^* whenever extending the tape and increases $\#A$ by an appropriate amount $\Theta(3^{s_{ct}})$ on every step.

How fast is the simulation with these parameters? From Section 6.7.1 we know that the expected time per microstep is $O(1/\lambda) = O(\frac{v(1/\delta + s_{ct0} + i)^4}{k3^{s_{ct}}})$. Since the total molecular count is asymptotically

* If $i_0 > 1/\delta + 1$, then $\delta > \int_{i_0-1}^{\infty} \frac{1}{x^2} dx > \sum_{x=i_0}^{\infty} \frac{1}{x^2}$.

$O(\#A) = O((1/\delta + s_{ct0} + i)3^{s_{ct}})$, this expected time is $O((1/\delta + s_{ct0} + i)^5/k)$. However, unlike in the bounded time/space simulations and the unbounded RM simulation, this expected time is conditional on all the previous microsteps being correct because if a microstep is incorrect, A and A^* may increase by an incorrect amount (for example reactions tripling $\#A$ akin to $A \rightarrow A^\dagger$ and $A^\dagger \rightarrow 3A$ can drive A arbitrarily high if the catalyst state species for both reactions are erroneously present simultaneously). Nonetheless, the expected duration of a microstep conditional on the entire simulation being correct is at most a factor of $1/(1 - \delta)$ larger than this.* Since we can assume δ will always be bounded above by a constant less than one, the expected duration of a microstep conditional on the entire simulation being correct is still $O((1/\delta + s_{ct0} + i)^5/k)$. By Lemma 6.7.9, this yields total expected time to simulate t_{tm} steps of a TM using at most s_{tm} space and with initial input of size s_{tm0} is $O((1/\delta + s_{tm0} + t_{tm}s_{tm})^5 t_{tm}s_{tm}/k)$ assuming the entire simulation is correct.

6.7.6 Decidability of Reachability

We reduce the reachability question in SCRN to the reachability question in Vector Addition Systems (VAS), a model of asynchronous parallel processes developed by Karp and Miller [18]. In the VAS model, we consider walks through a p dimensional integer lattice, where each step must be one of a finite set of vectors in \mathbb{N}^p , and each point in the walk must have no negative coordinates. It is known that the following reachability question is decidable: given points \mathbf{x} and \mathbf{y} , is there a walk that reaches some point $\mathbf{y}' \geq \mathbf{y}$ from \mathbf{x} [18]? The correspondence between VASs and SCRN is straightforward [7]. First consider chemical reactions in which no species occurs both as a reactant and as a product (i.e., reactions that have no catalysts). When such a reaction $\alpha = \langle \mathbf{l}, \mathbf{r}, k \rangle$ occurs, the state of the SCRN changes by addition of the vector $-\mathbf{l} + \mathbf{r}$. Thus the trajectory of states is a walk through \mathbb{N}^p wherein each step is any of a finite number of reactions, subject to the constraint requiring that the number of molecules of each species remain non-negative. Karp and Miller’s decidability results for VASs then directly imply that our reachability question of whether we ever enter a state greater than or equal to some target state is decidable for catalyst-free SCRN. The restriction to catalyst-free reactions is easily lifted: each catalytic reaction can be replaced by two new reactions involving a new molecular species after which all reachability questions (not involving the new species) are identical for the catalyst-free and the catalyst-containing networks.

Acknowledgments

We thank G. Zavattaro for pointing out an error in an earlier version of this manuscript. This work is supported in part by the “Alpha Project” at the Center for Genomic Experimentation and Computation, an NIH Center of Excellence (Grant No. P50 HG02370), as well as NSF Grant No. 0523761 and NIMH Training Grant MH19138-15.

Bibliography

- [1] D. Adalsteinsson, D. McMillen, and T. C. Elston. Biochemical network stochastic simulator (BioNetS): software for stochastic modeling of biochemical networks. *BMC Bioinformatics*, 5, 2004.
- [2] D. Angluin, J. Aspnes, and D. Eisenstat. Fast computation by population protocols with a leader. Technical report, Yale University, 2006. (Extended abstract to appear, DISC 2006.).
- [3] A. P. Arkin, J. Ross, and H. H. McAdams. Stochastic kinetic analysis of a developmental pathway bifurcation in phage- λ Escherichia coli. *Genetics*, 149:1633–1648, 1998.
- [4] B. Barak. A probabilistic-time hierarchy theorem for ‘slightly non-uniform’ algorithms. In *Proceedings of the International Workshop on Randomization and Computation*, 2002.

*This follows from the fact that $\mathbb{E}[X|A] \leq (1/\Pr[A])\mathbb{E}[X]$ for random variable X and event A , and that the expected microstep duration conditional on the previous and current microsteps being correct is the same as the expected microstep duration conditional on the entire simulation being correct.

- [5] C. H. Bennett. The thermodynamics of computation — a review. *International Journal of Theoretical Physics*, 21(12):905–939, 1982.
- [6] G. Berry and G. Boudol. The chemical abstract machine. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Annual Symposium on Principles of Programming Languages*, pages 81–94, 1990.
- [7] M. Cook. *Networks of Relations*. PhD thesis, California Institute of Technology, 2005.
- [8] A. P. de Silva and N. D. McClenaghan. Molecular-scale logic gates. *Chemistry — A European Journal*, 10(3):574–586, 2004.
- [9] M. B. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403:335–338, 2000.
- [10] M. B. Elowitz, A. J. Levine, E. D. Siggia, and P. S. Swain. Stochastic gene expression in a single cell. *Science*, 297:1183–1185, 2002.
- [11] P. Érdi and J. Tóth. *Mathematical Models of Chemical Reactions : Theory and Applications of Deterministic and Stochastic Models*. Manchester University Press, 1989.
- [12] S. N. Ethier and T. G. Kurtz. *Markov Processes: Characterization and Convergence*. John Wiley & Sons, 1986.
- [13] M. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *Journal of Physical Chemistry A*, 104:1876–1889, 2000.
- [14] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81:2340–2361, 1977.
- [15] D. T. Gillespie. A rigorous derivation of the chemical master equation. *Physica A*, 188:404–425, 1992.
- [16] D. T. Gillespie. Stochastic simulation of chemical kinetics. *Annual Review of Physical Chemistry*, 58:35–55, 2007.
- [17] P. Guptasarma. Does replication-induced transcription regulate synthesis of the myriad low copy number proteins of Escherichia coli? *Bioessays*, 17:987–997, 1995.
- [18] R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(4):147–195, 1969.
- [19] A. M. Kierzek. STOCKS: STOChastic kinetic simulations of biochemical systems with Gillespie algorithm. *Bioinformatics*, 18:470–481, 2002.
- [20] T. G. Kurtz. The relationship between stochastic and deterministic models for chemical reactions. *The Journal of Chemical Physics*, 57:2976–2978, 1972.
- [21] B. Levin. *Genes VII*. Oxford University Press, 1999.
- [22] A. M. L. Liekens and C. T. Fernando. Turing complete catalytic particle computers. In *Proceedings of Unconventional Computing Conference*, 2006.
- [23] J. Macdonald, Y. Li, M. Sutovic, H. Lederman, K. Pendri, W. Lu, B. L. Andrews, D. Stefanovic, and M. N. Stojanovic. Medium scale integration of molecular logic gates in an automaton. *Nano Letters*, 6:2598–2603, 2006.
- [24] M. O. Magnasco. Chemical kinetics is Turing universal. *Physical Review Letters*, 78:1190–1193, 1997.
- [25] H. H. McAdams and A. P. Arkin. Stochastic mechanisms in gene expression. *Proceedings of the National Academy of Sciences*, 94:814–819, 1997.
- [26] D. A. McQuarrie. Stochastic approach to chemical kinetics. *Journal of Applied Probability*, 4:413–478, 1967.

- [27] M. L. Minsky. Recursive unsolvability of Post's Problem of 'tag' and other topics in theory of Turing machines. *Annals of Math*, 74:437–455, 1961.
- [28] T. Neary and D. Woods. A small fast universal Turing machine. Technical report, National University of Maynooth, 2005.
- [29] G. Paun and G. Rozenberg. A guide to membrane computing. *Theoretical Computer Science*, 287:73–100, 2002.
- [30] P. W. Rothemund. A DNA and restriction enzyme implementation of Turing machines. In *DNA-Based Computers*, pages 75–120, 1996.
- [31] P. W. Rothemund, N. Papadakis, and E. Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology*, 2:e424, 2004.
- [32] G. Seelig, D. Soloveichik, D. Y. Zhang, and E. Winfree. Enzyme-free nucleic acid logic circuits. *Science*, 314:1585–1588, 2006.
- [33] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing, 1997.
- [34] D. Sprinzak and M. B. Elowitz. Reconstruction of genetic circuits. *Nature*, 438:443–448, 2005.
- [35] Stochastic simulation implementations. Systems Biology Workbench: <http://sbw.sourceforge.net>; BioSpice: <http://biospice.lbl.gov>; Stochastirator: <http://opnsr.bio.molsci.org>; STOCKS: <http://www.sysbio.pl/stocks>; BioNetS: <http://x.amath.unc.edu:16080/BioNetS>; SimBiology package for MATLAB: <http://www.mathworks.com/products/simbiology/index.html>.
- [36] M. N. Stojanovic, T. E. Mitchell, and D. Stefanovic. Deoxyribozyme-based logic gates. *Journal of the American Chemical Society*, 124:3555–3561, 2002.
- [37] G. M. Suel, J. Garcia-Ojalvo, L. M. Liberman, and M. B. Elowitz. An excitable gene regulatory circuit induces transient cellular differentiation. *Nature*, 440:545–550, 2006.
- [38] N. van Kampen. *Stochastic Processes in Physics and Chemistry*. Elsevier, revised edition edition, 1997.
- [39] K. Vasudeva and U. S. Bhalla. Adaptive stochastic-deterministic chemical kinetic simulations. *Bioinformatics*, 20:78–84, 2004.

Chapter 7

Robust Stochastic Chemical Reaction Networks

7.1 Abstract

The behavior of some stochastic chemical reaction networks is largely unaffected by slight inaccuracies in reaction rates. We formalize the robustness of state probabilities to reaction rate deviations, and describe a formal connection between robustness and efficiency of simulation. Without robustness guarantees, stochastic simulation seems to require computational time proportional to the total number of reaction events. Even if the concentration (molecular count per volume) stays bounded, the number of reaction events can be linear in the duration of simulated time and total molecular count. We show that the behavior of robust systems can be predicted such that the computational work scales linearly with the duration of simulated time and concentration, and only polylogarithmically in the total molecular count. Thus our asymptotic analysis captures the dramatic speed-up when molecular counts are large, and shows that for bounded concentrations the computation time is essentially invariant with molecular count. Finally, by noticing that even robust stochastic chemical reaction networks are capable of embedding complex computational problems, we argue that the linear dependence on simulated time and concentration is optimal.

7.2 Introduction

The stochastic chemical reaction network (SCRN) model of chemical kinetics is used in chemistry, physics, and computational biology. It describes interactions involving integer number of molecules as Markov jump processes [26, 33, 9, 14], and is used in domains where the traditional model of deterministic continuous mass action kinetics is invalid due to small molecular counts. Small molecular counts are prevalent in biology: for example, over 80% of the genes in the *E. coli* chromosome are expressed at fewer than a hundred copies per cell, with some key control factors present in quantities under a dozen [18, 23]. Indeed, experimental observations and computer simulations have confirmed that stochastic effects can be physiologically significant [25, 8, 32]. Consequently, the stochastic model is widely employed for modeling cellular processes (e.g., [5]) and is included in numerous software packages [34, 21, 1].* The stochastic model becomes equivalent to the classical law of mass action when the molecular counts of all participating species are large [22, 10].

Gillespie's stochastic simulation algorithm (SSA) can be used to model the behavior of SCRN [13]. However, simulation of systems of interest often requires an unfeasible amount of computational time. Some work has focused on optimizing simulation of large SCRN (many different species and reaction channels) with few reaction occurrences. For example, one can implement tricks to improve the speed of deciding which reaction occurs next among the many possible choices (e.g., [12]). However, for the purposes of this

*Some stochastic simulation implementations on the web: Systems Biology Workbench: <http://sbw.sourceforge.net>; BioSpice: <http://biospice.lbl.gov>; Stochastirator: <http://opnsrbcio.molsci.org>; STOCKS: <http://www.sysbio.pl/stocks>; BioNetS: <http://x.amath.unc.edu:16080/BioNetS>; SimBiology package for MATLAB: <http://www.mathworks.com/products/simbiology/index.html>

paper we suppose that the number of species and reactions is relatively small, and that it is fundamentally the number of reaction occurrences in a given interval of time that presents the difficulty. Because SSA simulates every single reaction event, simulation is slow when the number of reaction events is large.

On the face of it, simulation should be possible without explicitly modeling every reaction occurrence. In the mass action limit, fast simulation is achieved using numerical ODE solvers. The complexity of the simulation does not scale at all with the actual number of reaction occurrences but with overall simulation time and the concentration of the species. If the volume gets larger without a significant increase in concentration, mass action ODE solvers achieve a profound difference in computation time compared to SSA.* Moreover maximum concentration is essentially always bounded, because the model is only valid for solutions dilute enough to be well mixed, and ultimately because of the finite density of matter. However, mass action simulation can only be applied if molecular counts of *all* the species are large. Even one species that maintains a low molecular count and interacts with other species prevents the use of mass action ODE solvers.

Another reason why it seems that it should be possible to simulate stochastic chemical systems quickly, is that for many systems the behavior of interest does not depend crucially upon details of events. For example biochemical networks tend to be robust to variations in concentrations and kinetic parameters [27, 3]. If these systems are robust to many kinds of perturbations, including sloppiness in simulation, can we take advantage of this to speed up simulation? For example, can we approach the speed of ODEs but allow molecular counts of some species to be small? Indeed, tau-leaping algorithms (e.g., [15, 29, 7], see [17] for a review) are based on the idea that if we allow reaction propensities to remain constant for some amount of time τ , but therefore deviate slightly from their correct values, we don't have to explicitly simulate every reaction that occurs in this period of time (and can thus "leap" by amount of time τ).

In this paper we formally define robustness of the probability that the system is in a certain state at a certain time to perturbations in reaction propensities. We also provide a method for proving that certain simple systems are robust. We then describe a new approximate stochastic simulation algorithm called bounded tau-leaping (BTL), which naturally follows from our definition of robustness. In contrast to Gillespie's and others' versions of tau-leaping, in each step of our algorithm the leap time, rather than being a function of the current state, is a random variable. This algorithm naturally avoids some pitfalls of tau-leaping: the concentrations cannot become negative, and the algorithm scales to SSA when necessary, in a way that there is always at least one reaction per leap. However, in the cases when there are "opposing reactions" (canceling or partially cancelling each other) other forms of tau-leaping may be significantly faster (e.g., [28]).

BTL seems more amenable to theoretical analysis than Gillespie's versions [15, 16, 7], and may thus act as a stand-in for approximate simulation algorithms in analytic investigations. In this paper we use the language and tools of computational complexity theory to formally study how the number of leaps that BTL takes varies with the maximum molecular count m , time span of the simulation t , and volume V . In line with the basic computational complexity paradigm, our analysis is asymptotic and worst-case. "Asymptotic" means that we do not evaluate the exact number of leaps but rather look at the functional form of the dependence of their number on m , t , and V . This is easier to derive and allows for making fundamental distinctions (e.g., an exponential function is fundamentally larger than a polynomial function) without getting lost in the details. "Worst-case" means that we will not study the behavior of our algorithm on any particular chemical system but rather upper bound the number of leaps our algorithm takes independent of the chemical system. This will allow us to know that no matter what the system we are trying to simulate, it will not be worse than our bound.

In this computational complexity paradigm, we show that indeed robustness helps. We prove an upper bound on the number of steps our algorithm takes that is logarithmic in m , and linear in t and total concentration $C = m/V$. This can be contrasted with the exact SSA algorithm which, in the worst case, takes a number of steps that is linear in m , t , and C . Since a logarithmic dependence is much smaller than a linear one, BTL is provably "closer" to the speed of ODE solvers which have no dependence on m .[†]

Finally we ask whether it is possible to improve upon BTL, or did we exhaust the speed gains that can be obtained by using robustness? In the last section of the paper we connect this question to a widely held conjecture in computer science. Assuming the conjecture is true, we prove that there are robust systems

*As an illustrative example, a prokaryotic cell and a eukaryotic cell have similar concentrations of proteins but vastly different volumes.

[†]Indeed, the total molecular count m can be extremely large compared to its logarithm. For example, Avogadro's number = 6×10^{23} while its \log_2 is only 79.

whose behavior cannot be predicted in fewer computational steps than the number of leaps that BTL makes, ignoring multiplicative constant factors and powers of $\log m$. We believe other versions of tau-leaping have similar worst-case complexities as our algorithm, but proving equivalent results for them remains open.

7.3 Model and Definitions

A *Stochastic Chemical Reaction Network (SCRN)* \mathcal{S} specifies a set of N species S_i ($i \in \{1, \dots, N\}$) and M reactions R_j ($j \in \{1, \dots, M\}$). The *state* of \mathcal{S} is a vector $\vec{x} \in \mathbb{N}^N$ indicating the integral molecular counts of the species.* A reaction R_j specifies a reactants' stoichiometry vector $\vec{r}_j \in \mathbb{N}^N$, a products' stoichiometry vector $\vec{p}_j \in \mathbb{N}^N$, and a real-valued rate constant $k_j > 0$. We describe reaction stoichiometry using a standard chemical "arrow" notation; for example, if there are three species, the reaction $R_j: S_1 + S_2 \rightarrow S_1 + 2S_3$ has reactants vector $\vec{r}_j = (-1, -1, 0)$ and products vector $\vec{p}_j = (1, 0, 2)$. A reaction R_j is *possible* in state \vec{x} if there are enough reactant molecules: $(\forall i) x_i - r_{ij} \geq 0$. Then if reaction R_j occurs (or "fires") in state \vec{x} , the state changes to $\vec{x} + \vec{\nu}_j$, where $\vec{\nu}_j \in \mathbb{Z}^N$ is the state change vector for reaction R_j defined as $\vec{\nu}_j = \vec{p}_j - \vec{r}_j$. We follow Gillespie and others and allow unary ($S_i \rightarrow \dots$) and bimolecular ($2S_i \rightarrow \dots$ or $S_i + S_{i'} \rightarrow \dots$, $i \neq i'$) reactions only. Sometimes the model is extended to higher-order reactions [33], but the merit of this is a matter of some controversy.

Let us fix an SCRN \mathcal{S} . Given a starting state \vec{x}_0 and a fixed volume V , we can define a continuous-time Markov process we call an *SSA process*[†] \mathcal{C} of \mathcal{S} according to the following stochastic kinetics. Given a current state \vec{x} , the propensity function a_j of reaction R_j is defined so that $a_j(\vec{x})dt$ is the probability that one R_j reaction will occur in the next infinitesimal time interval $[t, t + dt)$. If R_j is a unimolecular reaction $S_i \rightarrow \dots$ then the propensity is proportional to the number of molecules of S_i currently present since each is equally likely to react in the next time instant; specifically, $a_j(\vec{x}) = k_j x_i$ for some reaction rate constant k_j . If R_j is a bimolecular reaction $S_i + S_{i'} \rightarrow \dots$, where $i \neq i'$, then the reaction propensity is proportional to $x_i x_{i'}$, which is the number of ways of choosing a molecule of S_i and a molecule of $S_{i'}$, since each pair is equally likely to react in the next time instant. Further, the probability that a particular pair reacts in the next time instant is inversely proportional to the volume, resulting in the propensity function $a_j(\vec{x}) = k_j \frac{x_i x_{i'}}{V}$. If R_j is a bimolecular reaction $2S_i \rightarrow \dots$ then the number of ways of choosing two molecules of S_i to react is $\frac{x_i(x_i-1)}{2}$, and the propensity function is $a_j(\vec{x}) = k_j \frac{x_i(x_i-1)}{2V}$.

Since the propensity function a_j of reaction R_j is defined so that $a_j(\vec{x})dt$ is the probability that one R_j reaction will occur in the next infinitesimal time interval $[t, t + dt)$, state transitions in the SSA process are equivalently described as follows: If the system is in state \vec{x} , no further reactions are possible if $\sum a_j(\vec{x}) = 0$. Otherwise, the time until the next reaction occurs is an exponential random variable with rate $\sum_j a_j(\vec{x})$. The probability that next reaction will be a particular R_{j^*} is $\alpha_{j^*}(\vec{x}) / \sum_j a_j(\vec{x})$.

We are interested in predicting the behavior of SSA processes. While there are potentially many different questions that we could be trying to answer, for simplicity we define the *prediction problem* as follows. Given an SSA process \mathcal{C} , a time t , a state \vec{x} , and $\delta \geq 0$, predict[‡] whether \mathcal{C} is in \vec{x} at time t , such that the probability that the prediction is incorrect is at most δ . In other words we are interested in algorithmically generating values of a Bernoulli random variable $I(\vec{x}, t)$ such that the probability that $I(\vec{x}, t) = 1$ when \mathcal{C} is not in \vec{x} at time t plus the probability that $I(\vec{x}, t) = 0$ when \mathcal{C} is in \vec{x} at time t is at most δ . We assume δ is some small positive constant. We can easily extend the prediction problem to a set of states Γ rather than a single target state \vec{x} by asking to predict whether the process is in any of the states in Γ at time t . Since Γ is meant to capture some qualitative feature of the SSA process that is of interest to us, it is called an *outcome*.

By decreasing the volume V (which speeds up all bimolecular reactions), increasing t , or allowing for more molecules (up to some bound m) we are increasing the number of reaction occurrences that we may need to consider. Thus for a fixed SCRN, one can try to upper bound the computational complexity of the prediction problem as a function of V , t , and m . Given a molecular count bound m , we define the *bounded-*

* $\mathbb{N} = \{0, 1, 2, \dots\}$ and $\mathbb{Z} = \{\dots, -1, 0, 1, \dots\}$.

[†]It is exactly the stochastic process simulated by Gillespie's Stochastic Simulation Algorithm (SSA) [13].

[‡]We phrase the prediction problem in terms appropriate for a simulation algorithm. An alternative formulation would be the problem of estimating the probability that the SSA process is in \vec{x} at time t . To be able to solve this problem using a simulation algorithm we can at most require that with probability at least δ_1 the estimate is within δ_2 of the true probability for some constants $\delta_1, \delta_2 > 0$. This can be attained by running the simulation algorithm a constant number of times.

count prediction problem as before, but allowing an arbitrary answer if the molecular count exceeds m within time t . Suppose \mathcal{P} is a bounded-count prediction problem with molecular count bound m , error bound δ , about time t and an SSA process in which the volume is V . We then say \mathcal{P} is a (m, t, C, δ) -prediction problem where $C = m/V$ is a bound on the maximum concentration.* Fixing some small δ , we study how the computational complexity of solving (m, t, C, δ) -prediction problems may scale with increasing m , t , and C . If the (m, t, C, δ) -prediction problem is regarding an outcome Γ consisting of multiple states, we require the problem of deciding whether a particular state is in Γ to be easily solvable. Specifically we require it to be solvable in time at most polylogarithmic in m .

It has been observed that permitting propensities to deviate slightly from their correct values, allows for much faster simulation, especially if the molecular counts of some species are large. This idea forms the basis of approximate stochastic simulation algorithms such as tau-leaping [15]. As opposed to the exact SSA process described above, consider letting the propensity function vary stochastically. Specifically, we define new propensity functions $a'_j(\vec{x}, t) = \xi_j(t)a_j(\vec{x})$ where $\{\xi_j(t)\}$ are random variables indexed by reaction and time. The value of $\xi_j(t)$ describes the deviation from the correct propensity of reaction R_j at time t , and should be close to 1. For any SSA process \mathcal{P} we can define a new stochastic process called a *perturbation* of \mathcal{P} through the choice of the distributions of $\{\xi_j(t)\}$. Note that the new process may not be Markov, and may not possess Poisson transition probabilities. If there is a $0 < \rho < 1$ such that $\forall j, t, (1 - \rho) \leq \xi_j(t) \leq (1 + \rho)$, then we call the new process a ρ -perturbation. There may be systems exhibiting behavior such that any slight inexactness in the calculation of propensities quickly gets amplified and results in qualitatively different behavior. However, for some processes, if ρ is a small constant, the ρ -perturbation may be a good approximation of the SSA process.

We now define our notion of robustness. Intuitively, we want the prediction problem to not be affected even if reaction propensities vary slightly. Formally, we say an SSA process \mathcal{C} is (ρ, δ) -robust with respect to state \vec{x} at time t if for any ρ -deviating process $\tilde{\mathcal{C}}$ based on \mathcal{C} , the probability of being in \vec{x} at time t is within plus or minus δ of the corresponding probability for \mathcal{C} . This definition can be extended to an outcome Γ similar to the definition on the prediction problem. Finally we say an SSA process \mathcal{C} is (ρ, δ) -robust with respect to a prediction problem (or bounded-count prediction problem) \mathcal{P} if \mathcal{C} is (ρ, δ) -robust with respect to the same state (or outcome) as specified in \mathcal{P} , at the same time as specified in \mathcal{P} .

For simplicity, we often use asymptotic notation. The notation $O(1)$ is used to denote an unspecified positive constant. This constant is potentially different every time the expression $O(1)$ appears.

7.4 Robustness Examples

In this section we elucidate our notion of robustness by considering some examples. In general, the question of whether a given SSA process is (ρ, δ) -robust for a particular outcome seems a difficult one. The problem is especially hard because we have to consider every possible ρ -perturbation — thus we may not even be able to give an approximate characterization of robustness by simulation with SSA. However, we can characterize the robustness of certain (simple) systems.

For an SSA process or ρ -perturbation \mathcal{C} , and outcome Γ , let $F^\Gamma(\mathcal{C}, t)$ be the probability of being in Γ at time t . Consider the SCRNs shown in Fig. 7.1(a). We start with 300 molecules of S_1 and S_3 each, and are interested in the outcome Γ of having at least 150 molecules of S_4 . The dashed line with circles shows F for the correct SSA process \mathcal{C} . (All plots of F are estimated from 10^3 SSA runs.) The two dashed lines without circles show F for two “extremal” ρ -perturbations: $\tilde{\mathcal{C}}^{+\rho}$ with constant $\xi_j(t) = 1 + \rho$, and $\tilde{\mathcal{C}}^{-\rho}$ with constant $\xi_j(t) = 1 - \rho$. What can we say about other ρ -perturbations, particularly where the $\xi_j(t)$ have much more complicated distributions? It turns out that for this SCRNs and Γ , we can prove that any ρ -perturbation falls within the bounds set by the two extremal ρ -perturbations $\tilde{\mathcal{C}}^{-\rho}$ and $\tilde{\mathcal{C}}^{+\rho}$. Thus F for any ρ -perturbation falls within the dashed lines. Formally, \mathcal{C} is monotonic with respect to Γ using the definition of monotonicity in Appendix 7.8.2. This is easily proven by Lemma 7.8.5 because every species is a reactant in at most one reaction. Then by Lemma 7.8.4, $F^\Gamma(\tilde{\mathcal{C}}^{-\rho}, t) \leq F^\Gamma(\tilde{\mathcal{C}}, t) \leq F^\Gamma(\tilde{\mathcal{C}}^{+\rho}, t)$ for any ρ -perturbation $\tilde{\mathcal{C}}$.

To see how the robustness of this system can be quantified using our definition of (ρ, δ) -robustness, first consider two time points $t = 4.5$ and $t = 6$. At $t = 4.5$, the probability that the correct SSA process \mathcal{C}

*Maximum concentration C is a more natural measure of complexity compared to V because similar to m and t , computational complexity increases as C increases.

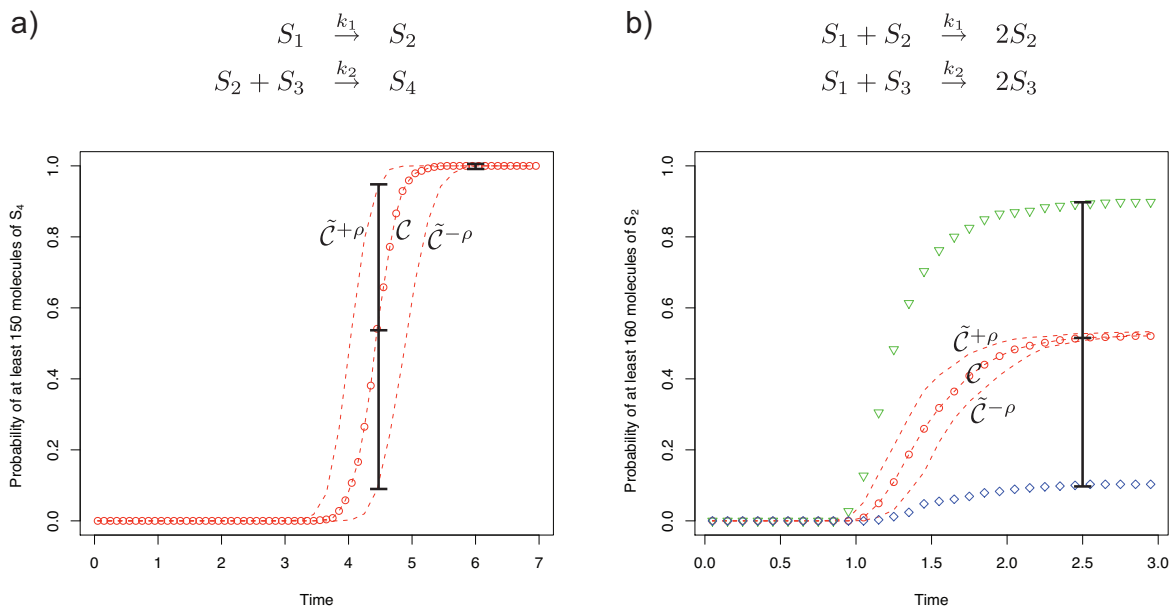


Figure 7.1: Examples of SCRN exhibiting contrasting degrees of robustness. The SSA process \mathcal{C} and outcome Γ are defined for the two systems by: (a) Rate constants: $k_1 = 1$, $k_2 = 0.001$; start state: $\vec{x}_0 = (300, 0, 300, 0)$; outcome Γ : $x_4 \geq 150$. (b) Rate constants: $k_1 = 0.01$, $k_2 = 0.01$; start state: $\vec{x}_0 = (300, 10, 10)$; outcome Γ : $x_2 \geq 160$. Plots show $F^\Gamma(\cdot, t)$ for an SSA process or ρ -perturbation estimated from 10^3 SSA runs. (Dashed line with circles) Original SSA process \mathcal{C} . (Dashed lines without circles) The two extremal ρ -perturbations: $\tilde{\mathcal{C}}^{+\rho}$ with constant $\xi_j(t) = 1 + \rho$, and $\tilde{\mathcal{C}}^{-\rho}$ with constant $\xi_j(t) = 1 - \rho$. For SCRN (b) we also plot $F^\Gamma(\cdot, t)$ for a ρ -perturbation with constant $\xi_1(t) = 1 + \rho$, $\xi_2(t) = 1 - \rho$ (triangles), or constant $\xi_1(t) = 1 - \rho$, $\xi_2(t) = 1 + \rho$ (diamonds). Perturbation parameter $\rho = 0.1$ throughout.

has produced at least 150 molecules of S_4 is slightly more than 0.5. The corresponding probability for ρ -perturbations of \mathcal{C} can be no larger than about 0.95 and no smaller than about 0.1. Thus \mathcal{C} is (ρ, δ) -robust with respect to outcome Γ at time $t = 4.5$ for $\rho = 0.1$ and δ approximately 0.45, but not for smaller δ . On the other hand at $t = 6$, the dashed lines are essentially on top of each other, resulting in a tiny δ . In fact δ is small for all times less than approximately 3.5 or greater than approximately 5.5.

What information did we need to be able to measure (ρ, δ) -robustness? Processes $\tilde{\mathcal{C}}^{-\rho}$ and $\tilde{\mathcal{C}}^{+\rho}$ are simply \mathcal{C} scaled in time. Thus knowing how $F^\Gamma(\mathcal{C}, t)$ varies with t allows one to quantify (ρ, δ) -robustness at the various times; $F^\Gamma(\mathcal{C}, t)$ can be estimated from multiple SSA runs of \mathcal{C} as in Fig. 7.1. Intuitively, \mathcal{C} is (ρ, δ) -robust for small δ at all times t when $F^\Gamma(\mathcal{C}, t)$ does not change quickly with t (see Appendix 7.8.2). For systems that are not monotonic, knowing how $F^\Gamma(\mathcal{C}, t)$ varies with time may not help with evaluating (ρ, δ) -robustness.

For a contrasting example, consider the SCRN in Fig. 7.1(b). We start with 300 molecules of S_1 , 10 molecules of S_2 , and 10 molecules of S_3 , and we are interested in the outcome of having at least 160 molecules of S_2 . Since S_1 is a reactant in both reactions, Lemma 7.8.5 cannot be used. In fact, the figure shows two ρ -perturbations (triangles and diamonds) that clearly escape from the boundaries set by the dashed lines. The triangles show F for the ρ -perturbation where the first reaction is maximally sped up and the second reaction is maximally slowed down. (Vice versa for the diamonds.) For characterization of the robustness of this system via (ρ, δ) -robustness, consider the time point $t = 2.5$. The probability of having at least 160 molecules of S_2 in the correct SSA process \mathcal{C} is around 0.5. However, this probability for ρ -perturbations of \mathcal{C} can deviate by at least approximately 0.4 upward and downward as seen by the two ρ -perturbations (triangles and diamonds). Thus at this time the system is not (ρ, δ) -robust for δ approximately 0.4. What about other ρ -deviations? It turns out that for this particular system, the two ρ -perturbations corresponding to the triangles and diamonds bound F in the same way that $\tilde{\mathcal{C}}^{-\rho}$ and $\tilde{\mathcal{C}}^{+\rho}$ bounded F in the first example (exercise left to the reader). Nonetheless, for general systems that are not monotonic it is not clear

how one can find such bounding ρ -perturbation and in fact they likely would not exist.

Of course, there are other types of SSA process that are not like either of the above examples: e.g., systems that are robust at many times but not monotonic. General ways of evaluating robustness of such systems remains an important open problem.

Finally, it is important to note that quantifying the robustness of SSA processes, even monotonic ones, seems to require computing many SSA runs. This is self-defeating when in practice one wants to show that the given SSA process is (ρ, δ) -robust in order to justify the use of an approximate simulation algorithm to quickly simulate it. In these cases, we have to consider (ρ, δ) -robustness a theoretical notion only.

7.5 Bounded Tau-Leaping

7.5.1 The Algorithm

We argued in the Introduction that sloppiness can allow for faster simulation. In this section we give a new approximate stochastic simulation algorithm called *bounded tau-leaping* (BTL) that simulates exactly a certain ρ -perturbation rather than the original SSA process. Consequently, the algorithm solves the prediction problem with allowed error δ for (ρ, δ) -robust SSA processes.

The algorithm is a variant of existing tau-leaping algorithms [17]. However, while other tau-leaping algorithms have an implicit notion of robustness, BTL is formally compatible with our explicit definition. As we'll see below, our algorithm also has certain other advantages over many previous tau-leaping implementations: it naturally disallows negative concentrations, and seems easier to formally analyze. In fact obtaining a result similar to Theorem 7.5.1 is an open question for other tau-leaping variants.

BTL has overall form typical of tau-leaping algorithms. Rather than simulating every reaction occurrence explicitly as per the SSA, BTL divides the simulation into leaps which group multiple reaction events. The propensities of all of the reactions are assumed to be fixed throughout the leap. This is obviously an approximation since each reaction event affects molecular counts and therefore the propensities. However, this approximation is useful because simulating the system with the assumption that propensities are fixed turns out to be much easier. Instead of having to draw random variables for each reaction occurrence, the number of random variables drawn to determine how many reaction firings occurred in a leap is independent of the number of reaction firings. Thus we effectively "leap" over all of the reactions within a leap in few computational steps. If molecular counts do not change by much within a leap then the fixed propensities are close to their correct SSA values and the approximation is good.

Our definition of a ρ -perturbation allows us to formally define "good." We want to guarantee that the approximate SSA process that tau-leaping actually simulates is a ρ -perturbation of the exact SSA process. We can achieve this as follows. If \vec{x} is the state on which the leap started, throughout the leap the simulated reaction propensities are fixed at their SSA propensities on x : $a_j(\vec{x})$. For any state \vec{y} within the leap we want the correct SSA propensities $a_j(\vec{y})$ to satisfy the following ρ -perturbation constraint ($0 < \rho < 1$): $(1 - \rho)a_j(\vec{y}) \leq a_j(\vec{x}) \leq (1 + \rho)a_j(\vec{y})$. As soon as we reach a state \vec{y} for which this constraint is violated, we start a new leap at \vec{y} which will use simulated reaction propensities fixed at $a_j(\vec{y})$. This ensures that at any time in the simulation, there is some $(1 - \rho) \leq \xi_j(t) \leq (1 + \rho)$ such that multiplying the correct SSA propensity of reaction R_j by $\xi_j(t)$ yields the propensity of R_j that the simulation algorithm is actually using. Therefore, we actually simulate a ρ -perturbation, and for (ρ, δ) -robust SSA processes, the algorithm can be used to provably solve the prediction problem with error δ .

Can we implement this simulation quickly, and, as promised, do little computation per leap? Note that in order to limit the maximum propensity deviation in a leap, we need to make the leap duration be a random variable dependent upon the stochastic events in the leap. If we evaluate $a_j(\vec{y})$ after each reaction occurrence in a leap to verify the satisfaction of the ρ -perturbation constraint, we do not save time over SSA. However, we can avoid this by using a stricter constraint we call the $\{\varepsilon_{ij}\}$ -perturbation constraint ($0 < \varepsilon_{ij} < 1$), defined as follows. If the leap starts in state \vec{x} , reaction R_j is allowed to change the molecular count of species S_i by at most plus or minus $\varepsilon_{ij}x_i$ within a leap. Again, as soon as we reach a state \vec{y} where this constraint is violated, we start a new leap at \vec{y} .

For any ρ , we can find a set of $\{\varepsilon_{ij}\}$ bounds such that satisfying the $\{\varepsilon_{ij}\}$ -perturbation constraint satisfies the ρ -perturbation constraint. For simplicity, suppose we set all ε_{ij} equal to some global ε . The maximum

0. Initialize with time $t = t_0$ and the system's state $\vec{x} = \vec{x}_0$.
1. With the system in state \vec{x} at time t , evaluate all the propensities a_j , and determine firing bounds b_j for all possible reactions, where b_j is the smallest positive integer such that $|b_j \nu_{ij}| > \varepsilon_{ij} x_i$ for some S_i .
2. Generate violating times $\tau_j \sim \text{Gamma}(b_j, a_j)$ for all possible reactions.
3. Find the first-violating reaction and set the step size to the time of the first violation: let $j^* = \text{argmin}_j \{\tau_j\}$ and $\tau = \tau_{j^*}$.
4. Determine the number of times each possible reaction occurred in interval τ : for $j \neq j^*$, $n_j \sim \text{Binomial}(b_j - 1, \tau/\tau_j)$; for j^* , $n_{j^*} = b_{j^*}$.
5. Effect the leap by replacing $t \leftarrow t + \tau$ and $\vec{x} \leftarrow \vec{x} + \sum_j \vec{\nu}_j n_j$.
6. Record (\vec{x}, t) as desired. Return to Step 1, or else end the simulation.

Figure 7.2: The bounded tau-leaping (BTL) algorithm. The algorithm is given the SCRN, the initial state \vec{x}_0 , the volume V , and a set of perturbation bounds $\{\varepsilon_{ij}\} > 0$. If the state at a specific time t_f is desired, the algorithm checks if $t + \tau > t_f$ in step (3), and if so uses $\tau = t_f - t$, and treats all reactions as not first-violating in step (4). $\text{Gamma}(n, \lambda)$ is a gamma distribution with shape parameter n and rate parameter λ . $\text{Binomial}(n, p)$ is a binomial distribution with number of trials n and success probability p .

change of any species S_i is plus or minus $M\varepsilon x_i$, where M is the number of reactions in the SCRN. We want to find an $\varepsilon > 0$ such that if the changes to all species stay within the $M\varepsilon$ bounds, then no reaction violates the ρ -perturbation constraint. Let us consider the most difficult case first which is a bimolecular reaction $R_j: 2S_i \rightarrow \dots$. The algorithm simulates its propensity as $a_j(\vec{x}) = k_j x_i(x_i - 1)/V$ throughout the leap. If $x_i < 2$, then $a_j(\vec{x}) = 0$, and as long as $M\varepsilon < 1$, $y_i < 2$ and $a_j(\vec{y}) = 0$, satisfying the ρ -perturbation constraint for R_j . Otherwise, suppose $x_i \geq 2$. At state \vec{y} within the leap, the SSA propensity is $a_j(\vec{y}) = k_j y_i(y_i - 1)/V \leq k_j(1 + M\varepsilon)x_i((1 + M\varepsilon)x_i - 1)/V$. So the left half of the ρ -perturbation constraint $a_j(\vec{x}) \leq (1 + \rho)a_j(\vec{y})$ is satisfied if $(1 - \rho)(1 + M\varepsilon)x_i((1 + M\varepsilon)x_i - 1) \leq x_i(x_i - 1)$. Similarly, $a_j(\vec{y}) = k_j y_i(y_i - 1)/V \geq k_j(1 - M\varepsilon)x_i((1 - M\varepsilon)x_i - 1)/V$ and the right half of the ρ -perturbation constraint $a_j(\vec{x}) \leq (1 + \rho)a_j(\vec{y})$ is satisfied if $(1 + \rho)(1 - M\varepsilon)x_i((1 - M\varepsilon)x_i - 1) \geq x_i(x_i - 1)$. These inequalities are satisfied for $x_i \geq 2$ when $\varepsilon \leq \frac{3}{4M} \left(1 - \sqrt{\frac{1+\rho/9}{1+\rho}}\right)$ (which also ensures that $M\varepsilon < 1$). It turns out this setting of ε also works for other reaction types,* and thus for any ρ we know how small ε needs to be such that satisfying the corresponding $\{\varepsilon_{ij}\}$ -perturbation constraint ensures that we are exactly simulating some ρ -perturbation.^{†‡}

Simulating a leap such that it satisfies the $\{\varepsilon_{ij}\}$ -perturbation constraint is easy and only requires drawing M gamma and $M - 1$ binomial random variables. Suppose the leap starts in state \vec{x} . For each reaction R_j , let b_j be the number of times R_j needs to fire to cause a violation of the $\{\varepsilon_{ij}\}$ bounds for some species. Thus b_j is the smallest positive integer such that $|b_j \nu_{ij}| > \varepsilon_{ij} x_i$ for some S_i . To determine τ , the duration of the leap, we do the following. First we determine when each reaction R_j would occur b_j times, by drawing from a gamma distribution with shape parameter b_j and rate parameter a_j . This generates a time τ_j for each reaction. The leap ends as soon as some reaction R_j occurs b_j times; thus to determine the duration of the

*Consider a unimolecular reaction $R_j: S_i \rightarrow \dots$. Using the same reasoning, the ρ -perturbation constraint for R_j is satisfied if $(1 - \rho)(1 + M\varepsilon)x_i \leq x_i$ and $(1 + \rho)(1 - M\varepsilon)x_i \geq x_i$. For any value of x_i , setting ε as stated fulfills these inequalities. Similarly, for a bimolecular reaction $R_j: S_i + S_{i'} \rightarrow \dots$, the inequalities are $(1 - \rho)(1 + M\varepsilon)^2 x_i x_{i'} \leq x_i x_{i'}$ and $(1 + \rho)(1 - M\varepsilon)^2 x_i x_{i'} \geq x_i x_{i'}$. Again for any $x_i, x_{i'}$, setting ε as stated fulfills these inequalities.

[†]Throughout the paper we assume that ρ, ε or $\{\varepsilon_{ij}\}$ are fixed and most of our asymptotic results do not show dependence on these parameters. Nonetheless, we can observe that for a fixed SCRN and for small enough ρ, ε can be within the range $O(1)\rho \leq \varepsilon \leq O(1)\rho$ and thus scales linearly with ρ . Therefore, in asymptotic results, the dependence on ε and ρ can be interchanged. Specifically, the ε dependence explored in Appendix 7.8.1 can be equally well expressed as a dependence on ρ .

[‡]Being given bounds in the form of $\{\varepsilon_{ij}\}$ rather than ρ allows some flexibility on the part of the user to assign less responsibility for a violation to a reaction that is expected to be fast compared to a reaction that is expected to be slow, thereby potentially speeding up the simulation, while still preserving the ρ -perturbation constraint. We do not explore this possibility further.

leap τ we take the minimum of the τ_j s. At this point we know that the first-violating reaction R_{j^*} — the one with the minimum τ_{j^*} — occurred b_{j^*} times. But we also need to know how many times the other reactions occur. Consider any other reaction R_j ($j \neq j^*$). Given that the b_j th occurrence of reaction R_j would have happened at time τ_j had the leap not ended, we need to distribute the other $b_j - 1$ occurrences to determine how many happen before time τ . The number of occurrences at time τ is given by the binomial distribution with number of trials $b_j(\vec{x}) - 1$ and success probability τ/τ_j . This enables us to define BTL as shown in Fig. 7.2.

The algorithm is called “bounded” tau-leaping because the deviations of reaction propensities within a leap are always bounded according to ρ . This is in contrast with other tau-leaping algorithms, such as Gillespie’s [7], in which the deviations in reaction propensities are small with high probability, but not always, and in fact can get arbitrarily high if the simulation is long enough. This allows BTL to satisfy our definition of a ρ -perturbation, and permits easier analysis of the behavior of the algorithm (see next section).

As any algorithm exactly simulating a ρ -perturbation would, BTL naturally avoids negative concentrations. Negative counts can occur only if an impossible reaction happens — in some state \vec{x} reaction R_j fires for which $a_j(\vec{x}) = 0$. But since in a ρ -perturbation propensity deviations are multiplicative, in state \vec{x} , $a'_j(\vec{x}, t) = \xi_j(t)a_j(\vec{x}) = 0$ and so R_j cannot occur.

On the negative side, in certain cases the BTL algorithm can take many more leaps than Gillespie’s tau-leaping [15, 16, 7] and other versions. Consider the case where there are two fast reactions that partially undo each others’ effect (for example the reactions may be reverses of each other). While both reactions may be occurring very rapidly, their propensities may be very similar (e.g., [28]). Gillespie’s tau-leaping will attempt to leap to a point where the molecular counts have changed enough according to the *averaged* behavior of these reactions. However, our algorithm considers each reaction separately and leaps to the point where the first reaction violates the bound on the change in a species in the absence of the other reactions. (Of course, the increased number of leaps that our algorithm takes results in greater accuracy, but this accuracy may be excessive).

7.5.2 Upper Bound on the Number of Leaps

Suppose we fix some SCRN of interest, and run BTL on different initial states, volumes, and lengths of simulated time. How does varying these parameters change the number of leaps taken by BTL? In this section, we prove that no matter what the SCRN is, we can upper bound the number of leaps as a function of the total simulated time t , the volume V , and the maximum total molecular count m encountered during the simulation. For simplicity we assume that all the ε_{ij} are equal to some global ε .*

Theorem 7.5.1. *For any SCRN \mathcal{S} with M species, any ε such that $0 < \varepsilon < 1/(12M)$, and any $\delta > 0$, there are constants $c_1, c_2, c_3 > 0$ such that for any bounds on time t and total molecular count m , for any volume V and any starting state, after $c_1 \log m + c_2 t (C + c_3)$ leaps where $C = m/V$, either the bound on time or the bound on total molecular count will be exceeded with probability at least $1 - \delta$.*

Proof. The proof is presented in Appendix 7.8.1. □

Note that the upper bound on ε implies that the algorithm is exactly simulating some ρ -perturbation (see previous section).

Intuitively, a key to the argument is that the propensity of a reaction decreasing a particular species is linear to the amount of that species (since the species must appear as a reactant). This allows us to bound the decrease of any species if a leap is short. Actually this implies that a short leap probably increases the amount of some species by a lot (some species must cause a violation — if not by a decrease it must be by an increase). This allows us to argue that if we have a lot of long leaps we exceed our time bound t and if we have a lot of short leaps we exceed our bound on total molecular count m . In fact because the effect of leaps is multiplicative, logarithmically many short leaps are enough to exceed m .

It is informative to compare this result with exact SSA, which in the worst case takes $O(1) m t (C + O(1))$ steps, since each reaction occurrence corresponds to an SSA step and the maximum reaction propensity is $k_j m^2/V$ or $k_j m$. Since m can be very large, the speed improvement can be profound.

*Alternatively, the theorem and proof can be easily changed to use min/max $\{\varepsilon_{ij}\}$ values where appropriate.

We believe, although it remains to be proven, that other versions of tau-leaping (see e.g., [17] for a review) achieve the same asymptotic worst case number of leaps as our algorithm.

How much computation is required per each leap? Each leap involves arithmetic operations on the molecular counts of the species, as well as drawing from a gamma and binomial distributions. Since there are fast algorithms for obtaining instances of gamma and binomial random variables (e.g., [2, 20]), we do not expect a leap of BTL to require much more computation than other forms of tau-leaping, and should not be a major contributor to the total running time. Precise bounds are dependent on the model of computation. (In the next section we state reasonable asymptotic bounds on the computation time per leap for a randomized Turing machine implementation of BTL.)

7.6 On the Computational Complexity of the Prediction Problem for Robust SSA Processes

What is the computational complexity inherent in the prediction problem for robust SSA processes, and how close does BTL come to the optimum computation time? In order to be able to consider these questions formally, we specify our model of computation as being randomized Turing machines. Then in terms of maximum total molecular count m , $\log m$ computation time is required to simply read in the initial state of the SSA process and target state of the prediction problem. We say that computation time polylogarithmic in m is *efficient in m* . What about the length of simulated time t and maximum concentration C ? We have shown that the number of leaps that BTL takes scales at most linearly with t and C . However, for some systems there are analytic shortcuts to determining the probability of being in Γ at time t . For instance the “exponential decay” SCRN consisting of the single reaction $S_1 \rightarrow S_2$ is easily solvable analytically [24]. The calculation of the probability of being in any given state at any given time t (among other questions) can be solved in time that grows minimally with t and C . In this section we prove that despite such examples, for any algorithm solving prediction problems for robust SSA processes, there are prediction problems about such processes that cannot be solved faster than linear in t and C , assuming a widely believed conjecture in computational complexity theory. We prove this result for any algorithm that is efficient in m . We finally argue, with certain caveats regarding implementing BTL on a Turing machine, that as an algorithm for solving prediction problems for robust SSA processes, BTL is asymptotically optimal among algorithms efficient in m because its computation time scales linearly with t and C .

In order to prove formal lower bounds on the computational complexity of the prediction problem, we must be specific about our computation model. We use the standard model of computation which captures stochastic behavior: randomized Turing machines (TM). A randomized TM is a non-deterministic TM* allowing multiple possible transitions at a point in a computation. The actual transition taken is uniform over the choices. (See for example [30] for equivalent formalizations.) We say a given TM on a given input runs in computational time t_{tm} if there is no set of random choices that makes the machine run longer.

We want to show that for some SCRNs, there is no method of solving the prediction problem fast, no matter how clever we are. We also want these stochastic processes to be robust despite having difficult prediction problems. We use the following two ideas. First, a method based on [4] shows that predicting the output of given randomized TMs can be done by solving a prediction problem for certain robust SSA processes, similar to the construction of Chapter 6. Second, an open conjecture, but one that is widely believed in computational complexity theory, bounds how quickly the output of randomized TMs can be determined.

Computational complexity theory concerns measuring how the computational resources required to solve a given problem scale with input size n (in bits). The two most prevalent efficiency measures are time and space — the number of TM steps and the length of the TM tape required to perform the computation. Let us say a Boolean function $f(x)$ is *probabilistically computable* by a TM M in time $t(|x|)$ and space $s(|x|)$ if $M(x)$ runs in time $t(|x|)$ using space at most $s(|x|)$, and with probability at least $2/3$ outputs $f(x)$.[†] It is widely believed[‡] that the following *hierarchy conjecture* is true:

*Arbitrary finite number of states and tapes. Without loss of generality, we can assume a binary alphabet.

[†]Any other constant probability bounded away from $1/2$ will do just as well: to achieve a larger constant probability of being correct, we can repeat the computation a constant number of times and take majority vote.

[‡]If we do not allow any chance of error and do not restrict space usage, the equivalent statement is proven as the (deterministic)

Conjecture 7.6.1 ((Probabilistic, Space-Limited) Time Hierarchy). *For any $\alpha < 1$, and polynomials $t(n)$ and $s(n)$ such that $t(n)^\alpha$ and $s(n)$ are at least linear, there are Boolean functions that can be probabilistically computed within time and space bounds $t(n)$ and $s(n)$, but not in time $O(1)t(n)^\alpha$, even allowing unrestricted space usage.*

Intuitively, we take a Boolean function that requires $t(n)$ time and embed it in a chemical system in such a way that solving the prediction problem is equivalent to probabilistically computing the function. The conjecture implies that we cannot solve the prediction problem fast enough to allow us to solve the computational problem faster than $t(n)$. Further, since the resulting SSA process is robust, the result lower-bounds the computational complexity of the prediction problem for robust processes. Note that we need a time hierarchy conjecture that restricts the space usage and talks about probabilistic computation because it is impossible to embed a TM computation in an SCRN such that its computation is error free (see Chapter 6) and such embedding seems to require more time as the space usage increases.

The following theorem lower-bounds the computational complexity of the prediction problem. The bound holds even if we restrict ourselves to robust processes. It shows that this computational complexity is at least linear in t and C , as long as the dependence on m is at most polylogarithmic. It leaves the possibility that there are algorithms for solving the prediction problem that require computation time more than polylogarithmic in m but less than linear in t or C . Let the prediction problem be specified by giving the SSA process (via the initial state and volume), the target time t , and the target outcome Γ in some standard encoding such that whether a state belongs to Γ can be computed in time polylogarithmic in m .

Theorem 7.6.1. *Fix any perturbation bound $\rho > 0$ and $\delta > 0$. Assuming the hierarchy conjecture (Conjecture 7.6.1), there is an SCRN \mathcal{S} such that for any prediction algorithm \mathcal{A} and constants $c_1, c_2, \beta, \eta, \gamma > 0$, there is an SSA process \mathcal{C} of \mathcal{S} and a $(m, t, C, 1/3)$ -prediction problem \mathcal{P} of \mathcal{C} such that \mathcal{C} is (ρ, δ) -robust with respect to \mathcal{P} , and \mathcal{A} cannot solve \mathcal{P} in computational time $c_1 (\log m)^\beta t^\eta (C + c_2)^\gamma$ if $\eta < 1$ or $\gamma < 1$.*

Proof. The proof is presented in Appendix 7.8.4. □

With the above theorem demarcating a boundary of what is possible, the natural question is how close to optimal does BTL come? In the previous section, we have derived an upper bound on the number of leaps that our algorithm takes. However, we need to address how the idealized bounded-tau leaping algorithm presented in Section 7.5.1 can be implemented on a randomized TM which allows only finite precision arithmetic and a restricted model of randomness generation. We have to deal with round-off error and approximate gamma and binomial random number generators, whose effect on the probability of outcome is difficult to track formally. Further, the computational complexity of these operations is a function of the bits of precision and is complicated to rigorously bound.

As shown in Appendix 7.8.5, BTL on a randomized TM runs in total computation time

$$O(1)((\log(m))^{O(1)} + l) t (C + O(1)) \tag{7.1}$$

where, in each leap, polylogarithmic time in m is required for arithmetic manipulation of molecular counts, and l captures the extra computation time required for the real number operations and drawing from the gamma and binomial distributions. l is potentially a function of m, V, t , and the bits of precision used. Assuming efficient methods for drawing the random variables, l is likely very small compared to the total number of leaps. Further, as we discuss in Appendix 7.8.5, assuming round-off errors and deviations due to approximate random number generation do not cause problems, for any fixed $\delta_1 < \delta$, the randomized TM implementation solves the (m, t, C, δ) -prediction problem for (ρ, δ_1) -robust processes. So in as far as l in (eq. 7.1) can be neglected, and further assuming we can ignore errors introduced due to finite precision arithmetic and approximate random number generation, bounded-tau leaping is asymptotically optimal up to multiplicative constants and powers of $\log m$ among algorithms efficient in m .

Assuming the hierarchy conjecture and with the caveats above, we have matching asymptotic upper and lower bounds in t and C for solving the prediction problem for robust SSA processes. However, non-robust systems may require much more computation time to solve the prediction problem since BTL may not be used. This may mean that there are ways to imbed computation in non-robust SSA processes that is more efficient than the method of [4] that is used in the proof of Theorem 7.6.1.

time hierarchy theorem [30]. Also see [6, 11] for progress in proving the probabilistic version with unrestricted space usage.

7.7 Discussion

The behavior of many stochastic chemical reaction networks does not depend crucially on getting the reaction propensities exactly right, prompting our definition of ρ -perturbations and (ρ, δ) -robustness. A ρ -perturbation of an SSA process is a stochastic process with stochastic deviations of the reaction propensities from their correct SSA values. These deviations are multiplicative and bounded between $1 - \rho$ and $1 + \rho$. If we are concerned with how likely the SSA process is in a given state at a given time, then (ρ, δ) -robustness captures how far these probabilities may deviate for a ρ -perturbation.

We formally showed that predicting the behavior of robust processes does not require simulation of every reaction event. Specifically, we described a new approximate simulation algorithm called bounded tau-leaping (BTL) that simulates a certain ρ -perturbation as opposed to the exact SSA process. The accuracy of the algorithm in making predictions about the state of the system at given times is guaranteed for (ρ, δ) -robust processes. We further proved an upper bound on the number of leaps of BTL that helps explain the savings over SSA. The bound is a function of the desired length of simulated time t , volume V , and maximum molecular count encountered m . This bound scales linearly with t and $C = m/V$, but only logarithmically with m , while the total number of reactions (and therefore SSA steps) may scale linearly with t , C , and m . When total concentration is limited, but the total molecular count is large, this represents a profound improvement over SSA. We also argue that asymptotically as a function of t and C our algorithm is optimal in as far as no algorithm can achieve sublinear dependence of the number of leaps on t or C . This result is proven based on a widely believed assumption in computational complexity theory. Unlike Gillespie’s tau-leaping [7], our algorithm seems better suited to theoretical analysis. Thus while we believe other versions of tau-leaping have similar worst-case running times, the results analogous to those we obtain for BTL remain to be proved.

Our results can also be seen to address the following question. If concerned solely with a particular outcome rather than with the entire process trajectory, can one always find certain shortcuts to determine the probability of the outcome without performing a full simulation? Since our lower bound on computation time scales linearly with t , it could be interpreted to mean that, except in problem-specific cases, there is no shorter route to predicting the outcomes of stochastic chemical processes than via simulation. This negative result holds even restricting to the class of robust SSA processes.

While the notion of robustness is a useful theoretical construct, how practical is our definition in deciding whether a given system is suitable to approximate simulation via BTL or not? We prove that for the class of monotonic SSA processes, robustness is guaranteed at all times when in the SSA process the outcome probability is stable over an interval of time determined by ρ . However, it is not clear how this stability can be determined without SSA simulation. Even worse, few systems of interest are monotonic. Consequently, it is compelling to develop techniques to establish robustness for more general classes of systems. A related question is whether it is possible to connect our notion of robustness to previously studied notions in mass action stability analysis [19, 31].

7.8 Appendix

7.8.1 Proof of Theorem 7.5.1: Upper Bound on the Number of Leaps

In this section we prove Theorem 7.5.1 from the text, which upper bounds the number of leaps BTL takes as a function of m , t , and C :

Theorem. *For any SCRN \mathcal{S} with M species, any ε such that $0 < \varepsilon < 1/(12M)$, and any $\delta > 0$, there are constants $c_1, c_2, c_3 > 0$ such that for any bounds on time t and total molecular count m , for any volume V and any starting state, after $c_1 \log m + c_2 t (C + c_3)$ leaps where $C = m/V$, either the bound on time or the bound on total molecular count will be exceeded with probability at least $1 - \delta$.*

We prove a more detailed bound than stated in the theorem above which explicitly shows the dependence on ε hidden in the constants. Also since we introduce the asymptotic results only the end of the argument, the interested reader may easily investigate the dependence of the constants on other parameters of the SCRN such as N , M , ν_{ij} , and k_j . We also show an approach to probability 1 that occurs exponentially fast as the bound increases: if the bound above evaluates to n , then the probability that the algorithm does not exceed m or t in n leaps is at most $2e^{-O(1)n}$.

Our argument starts with a couple of lemmas. Looking within a single leap, the first lemma bounds the decrease in the molecular count of a species due to a given reaction as a function of time. The argument is essentially that for a reaction to decrease the molecular count of a species, that species must be a reactant, and therefore the propensity of the reaction is proportional to its molecular count. Thus we see a similarity to an exponential decay process and use this to bound the decrease. Note that a similar result does not hold for the *increase* in the molecular count of a species, since the molecular count of the increasing species need not be in the propensity function.* Then the second lemma uses the upper bound on how fast a species can decrease (the first lemma), together with the fact that in a leap some reaction must change some species by a relatively large amount, to classify leaps into those that either (1) take a long time or (2) increase some species significantly without decreasing any other species by much. Finally we show that this implies that if there are too many leaps we either violate the time bound or the total molecular count bound.

For the following, values f and g will be free parameters to be determined later. It helps to think of them as $0 < f \ll g \ll 1$. How long does it take for a reaction to decrease x_i by g th fraction of the violation bound εx_i ? The number of occurrences of R_j to decrease x_i by $g\varepsilon x_i$ or more is at least $g\varepsilon x_i / |\nu_{ij}|$. The following lemma bounds the time required for these many occurrences to happen.

Lemma 7.8.1. *Take any f and g ($0 < f, g < 1$), any reaction R_j and species S_i such that $\nu_{ij} < 0$, any state \vec{x} , and any ε . Assuming that the propensity of R_j is fixed at $a_j(\vec{x})$, with probability at least $1 - f/g$, fewer than $g\varepsilon x_i / |\nu_{ij}|$ occurrences of R_j happen in time $f\varepsilon / (|\nu_{ij}| k_j)$ if R_j is unimolecular, or time $f\varepsilon / (|\nu_{ij}| k_j C)$ if R_j is bimolecular.*

Proof. For reaction R_j to decrease the amount of S_i , it must be that S_i is a reactant, and thus x_i is a factor in the propensity function. Suppose R_j is unimolecular. Then $a_j = k_j x_i$ and the expected number of occurrences of R_j in time $f \frac{\varepsilon}{|\nu_{ij}| k_j}$ is $a_j f \frac{\varepsilon}{|\nu_{ij}| k_j} \leq f \frac{\varepsilon x_i}{|\nu_{ij}|}$. The desired result then follows from Markov's inequality. If R_j is bimolecular with $S_i \neq S_{i'}$ being the other reactant then $a_j = k_j \frac{x_i x_{i'}}{V}$; alternatively, $a_j = k_j \frac{x_i(x_i-1)}{V}$ if R_j is bimolecular with identical reactants. In general for bimolecular reactions $a_j \leq k_j x_i C$. So the expected number of occurrences of R_j in time $f \frac{\varepsilon}{|\nu_{ij}| k_j C}$ is $a_j f \frac{\varepsilon}{|\nu_{ij}| k_j C} \leq f \frac{\varepsilon x_i}{|\nu_{ij}|}$. The desired result follows as before. \square

Let time $\tilde{\tau}$ be the minimum over all reactions R_j and S_i such that $\nu_{ij} < 0$ of $1/(|\nu_{ij}| k_j)$ if R_j is unimolecular, or $1/(|\nu_{ij}| k_j C)$ if R_j is bimolecular. We can think of $\tilde{\tau}$ setting the units of time for our argument. The above lemma implies that with probability at least $1 - f/g$ no reaction decreases x_i by $g\varepsilon x_i$ or more within time $f\varepsilon \tilde{\tau}$. The following lemma defines typical leaps; they are of two types: long or S_i -increasing. Recall M is the number of reaction channels and N is the number of species.

Lemma 7.8.2. (Typical leaps). *For any f and g ($0 < f, g < 1$), and for any ε , with probability at least $1 - NMf/g$ one of the following is true of a leap:*

1. (long leap) $\tau > f\varepsilon \tilde{\tau}$
2. (S_i -increasing leap) $\tau \leq f\varepsilon \tilde{\tau}$, and the leap increases some species S_i at least as $x_i \mapsto x_i + \lceil \varepsilon x_i \rceil - gM\varepsilon x_i$, while no species $S_{i'}$ decreases as much as $x_{i'} \mapsto x_{i'} - gM\varepsilon x_{i'}$.

Proof. By the union bound over the M reaction channels and the N species, Lemma 7.8.1 implies that the probability that *some* reaction decreases the amount of *some* species S_i by $g\varepsilon x_i$ or more in time $f\varepsilon \tilde{\tau}$ is at most NMf/g . Now suppose this unlucky event does not happen. Then if the leap time is $\tau \leq f\varepsilon \tilde{\tau}$, no decrease is enough to cause a violation of the deviation bounds, and thus it must be that some reaction R_j increases some species S_i by more than εx_i . (Since R_j must occur an integer number of times, it actually must increase S_i by $\lceil \varepsilon x_i \rceil$ or more.) Since no reaction decreases S_i by $g\varepsilon x_i$ or more, we can be sure that S_i increases at least by $\lceil \varepsilon x_i \rceil - gM\varepsilon x_i$. \square

Lemma 7.8.3. *For any species S_i , a leap decreases S_i at most as $x_i \mapsto x_i - M \lceil \varepsilon x_i \rceil - 2$.*

*If a reaction is converting a populous species to a rare one, the rate of the increase of the rare species can be proportional to m times its molecular count. The rate of decrease, however, is always proportional to the molecular count of the decreasing species, or proportional to C times the molecular count of the decreasing species (as we'll see below).

Proof. At most M reactions may be decreasing S_i . A reaction can decrease S_i by as much as $\lfloor \varepsilon x_i \rfloor$ without causing a violation of the deviation bounds. The last reaction firing that causes the violation of the deviation bounds ending the leap uses up at most 2 molecules of S_i (since reactions are at most bimolecular). \square

Note that a similar lemma does not hold for Gillespie’s tau-leaping algorithms [15, 16, 7] because the number of reaction firings in a leap can be only bounded probabilistically. With some small probability a leap can result in “catastrophic” changes to some molecular counts. Since with enough time such events are certain to occur, the asymptotic analysis must consider them. Consequently, asymptotic results analogous to those we derive in this section remain to be proved for tau-leaping algorithms other than BTL.

Our goal now is to use the above two lemmas to argue that if we have a lot of leaps, we would either violate the molecular count bound (due to many S_i -increasing leaps for the same S_i), or violate the time bound (due to long leaps). Let n be the total number of leaps. By Hoeffding’s inequality, with probability at least $1 - 2e^{-2n(NMf/g)^2}$ (i.e., exponentially approaching 1 with n), the total number of atypical steps is bounded as:

$$[\# \text{ of atypical leaps}] < 2nNMf/g. \quad (7.2)$$

Further, in order not to violate the time bound t , the number of long steps can be bounded as:

$$[\# \text{ of long leaps}] \leq t/(f\varepsilon\tilde{\tau}). \quad (7.3)$$

How can we bound the number of the other leaps (S_i -increasing, for some species S_i)? Our argument will be that having too many of such leaps results in an excessive increase of a certain species, thus violating the bound on the total molecular count. We start by choosing an S_i for which there is the largest number of S_i -increasing steps. Since there are N species, there must be a species S_i for which

$$[\# \text{ of } S_i\text{-increasing leaps}] > \frac{1}{N} \sum_{S_{i'} \neq S_i} [\# \text{ of } S_{i'}\text{-increasing leaps}]. \quad (7.4)$$

At this point, it helps to develop an alternative bit of notation labeling the different kinds of leaps with respect to the above-chosen species S_i to indicate how much x_i may change in the leap. Since our goal will be to argue that the molecular count of S_i must be large, we would like to lower-bound the increase in S_i and upper-bound the decrease. An atypical leap or a long leap we label “ $\downarrow\downarrow$ ”. By Lemma 7.8.3 these leaps decrease S_i at most as $x_i \mapsto x_i - M\lfloor \varepsilon x_i \rfloor - 2$. An S_i -increasing leap we label “ \uparrow ”. Finally, an $S_{i'}$ -increasing leap for $S_{i'} \neq S_i$ we label “ \downarrow ”. By Lemma 7.8.2, \uparrow leaps increase S_i at least as $x_i \mapsto x_i + \lfloor \varepsilon x_i \rfloor - gM\varepsilon x_i$, while \downarrow leaps decrease S_i by less than $x_i \mapsto x_i - gM\varepsilon x_i$.

We would like to express these operations purely in a multiplicative way so that they become commutative, allowing for bounding their total effect on x_i independent of the order in which these leaps occurred but solely as a function of the number of each type. Further, the multiplicative representation of the leap effects is important because we want to bound the number of leaps logarithmically in the maximum molecular count. Note that $\downarrow\downarrow$ leaps cause a problem because of the subtractive constant term, and \uparrow leaps cause a problem because if x_i drops to 0 multiplicative increases are futile. Nonetheless, for the sake of argument suppose we knew that throughout the simulation $x_i \geq 3$. Then assuming $\varepsilon \leq 1/(12M)$, we can bound the largest decrease due to a $\downarrow\downarrow$ leap multiplicatively as $x_i \mapsto (1/4)x_i$. Further, we lower-bound the increase due to a \uparrow leap as $x_i \mapsto (1 + (1 - gM)\varepsilon)x_i$. Then the lower bound on the final molecular count of S_i and therefore the total molecular count is

$$3(1 + (1 - gM)\varepsilon)^{n^\uparrow} (1 - gM\varepsilon)^{n^\downarrow} (1/4)^{n^{\downarrow\downarrow}} \leq m. \quad (7.5)$$

This implies an upper bound on the number of \uparrow leaps, that together with eqns. 7.2–7.4 provides an upper bound on the total number of leaps, as we’ll see below.

However, x_i might dip below 3 (including at the start of the simulation). We can adjust the effective number of \uparrow leaps to compensate for these dips. We say a leap is in a dip if it starts at $x_i < 3$. Observe that the first leap in a dip starts at $x_i < 3$ while the leap after a dip starts at $x_i \geq 3$. Thus, unless we end in a dip, cutting out the leaps in the dips can only decrease our lower bound on the final x_i . We’ll make an even looser bound and modify (7.5) simply by removing the contribution of the \uparrow leaps that are in dips.* How

*We know we cannot end in a dip if the resulting bound evaluates to 3 or more. Thus technically we assume $m \geq 3$ for the bound

many \uparrow leaps can be in dips? First let us ensure $g < 1/(3M)$. Then since a \downarrow leap decreases x_i by less than $gM\varepsilon x_i < x_i/3$, and the decrease amount must be an integer, a \downarrow leap cannot bring x_i below 3 starting at $x_i \geq 3$. Thus if we start at $x_i \geq 3$ a $\downarrow\downarrow$ leap must occur before we dip below 3. Thus the largest number of dips is $n^{\downarrow\downarrow} + 1$ (adding 1 since we may start the simulation below 3). Let n_d^\uparrow and $n_d^{\downarrow\downarrow}$ be the number of \uparrow and $\downarrow\downarrow$ leaps in the d th dip (we don't care about \downarrow leaps in a dip since they must leave x_i unchanged). Then $n_d^\uparrow < 2n_d^{\downarrow\downarrow} + 3$ and $\sum_d n_d^\uparrow < \sum_d 2n_d^{\downarrow\downarrow} + \sum_d 3 \leq 2n^{\downarrow\downarrow} + 3(n^{\downarrow\downarrow} + 1) = 5n^{\downarrow\downarrow} + 3$. Therefore, the adjusted bound (7.5) becomes: $3(1 + (1 - gM)\varepsilon)^{n^\uparrow - 5n^{\downarrow\downarrow} - 3}(1 - gM\varepsilon)^{n^\downarrow}(1/4)^{n^{\downarrow\downarrow}} \leq m$. For simplicity, we use the weaker bound

$$3(1 + (1 - gM)\varepsilon)^{n^\uparrow}(1 - gM\varepsilon)^{n^\downarrow}(1/4)^{6n^{\downarrow\downarrow} + 3} \leq m. \quad (7.6)$$

In order to argue that this bounds the number of \uparrow leaps, we need to make sure the \downarrow leaps and the $\downarrow\downarrow$ leaps don't cancel out the effect of the \uparrow leaps. By inequality 7.4 we know that $n^\downarrow < Nn^\uparrow$. If we can choose g to be a small enough constant such that more than N \downarrow leaps are required to cancel the effect of a \uparrow leap we would be certain the bound increases exponentially with n^\uparrow without caring about \downarrow leaps. Specifically, we choose a g small enough such that $(1 + (1 - gM)\varepsilon)(1 - gM\varepsilon)^N \geq 1 + \varepsilon/2$. For example we can let $g = \frac{1}{M}(1 - (9/10)^{1/N})$.^{*} Note that g depends only on constants N and M and is independent of ε . The bound then becomes $3(1 + \varepsilon/2)^{n^\uparrow}(1/4)^{6n^{\downarrow\downarrow} + 3}$.

Thus finally we have the following system of inequalities that are satisfied with probability exponentially approaching 1 as $n \rightarrow \infty$:

$$n = n^\uparrow + n^\downarrow + n^{\downarrow\downarrow} \quad (7.7)$$

$$n^{\downarrow\downarrow} \leq t/(f\varepsilon\tilde{\tau}) + 2nNMf/g \quad (7.8)$$

$$n^\downarrow < Nn^\uparrow \quad (7.9)$$

$$3(1 + \varepsilon/2)^{n^\uparrow}(1/4)^{6n^{\downarrow\downarrow} + 3} \leq m. \quad (7.10)$$

Solving for n we obtain[†]

$$n \leq \frac{h \log(m/3) + (12h + 1)t/(f\varepsilon\tilde{\tau}) + 6h}{(1 - 24hNMf/g)}$$

if $(1 - 24hf/g) > 0$ where $h = (N + 1)/\log(1 + \varepsilon/2)$ (also recall $g = \frac{1}{M}(1 - (9/10)^{1/N})$). To ensure this we let $f \leq g/(48hNM)$. Then with probability exponentially approaching 1 as n increases,

$$n \leq 2 \log(m/3) + 96(12h + 1)th/(g\varepsilon\tilde{\tau}) + 12h.$$

Asymptotically as $\varepsilon \rightarrow 0, m \rightarrow \infty, t \rightarrow \infty$ with the system of chemical equations being fixed, we have $g = O(1)$, $h \leq O(1)/\varepsilon$, and write the above as $n \leq O(1)(1/\varepsilon) \log m + O(1)(1/\varepsilon)^3 t/\tilde{\tau}$. Recall our unit of time $\tilde{\tau}$ was defined to be the minimum over all reactions R_j and species S_i such that $\nu_{ij} < 0$ of $1/(|\nu_{ij}|k_j)$ if R_j is unimolecular, or $1/(|\nu_{ij}|k_jC)$ if R_j is bimolecular. No matter what C is, we can say $\tilde{\tau} \geq 1/(O(1)C + O(1))$. Thus we can write the above as

$$n \leq O(1)(1/\varepsilon) \log m + O(1)(1/\varepsilon)^3 t(C + O(1)).$$

For any δ , we can find appropriate constants such that the above bound is satisfied with probability at least $1 - \delta$.

This bound on the number of leaps has been optimized for simplicity of proof rather than tightness. A more sophisticated analysis can likely significantly decrease the numerical constants. Further, we believe the cubic dependence on $1/\varepsilon$ in the time term is excessive.[‡]

to be always valid.

^{*}Since $g \leq 1/(3M)$, make the simplification $(1 + (1 - gM)\varepsilon) \geq (1 + 2\varepsilon/3)$ and solve for g . The solution is minimized when $\varepsilon = 1$.

[†]Logarithms are base-2.

[‡]The cubic dependence on $1/\varepsilon$ in the time term is due to having to decrease the probability of an atypical step as ε decreases. It may be possible to reduce the cubic dependence to a linear one by moving up the boundary between a dip and the multiplicative regime as a function of ε rather than fixing it at 3. The goal is to replace the constant base $(1/4)^{O(1)n^{\downarrow\downarrow} + O(1)}$ term with a $(1 -$

7.8.2 Proving Robustness by Monotonicity

In this section we develop a technique that can be used to prove the robustness of certain SSA processes. We use these results to prove the robustness of the example in Section 7.4 as well as of the construction of [4] simulating a Turing machine in Appendix 7.8.3.

Since ρ -perturbations are not Markovian, it is difficult to think about them. Can we use a property of the original SSA process that would allow us to prove robustness without referring to ρ -perturbations at all?

Some systems have the property that every reaction can only bring the system “closer” to the outcome of interest (or at least “no further”). Formally, we say an SSA process is *monotonic* for outcome Γ if for all reachable states \vec{x}, \vec{y} such that there is a reaction taking \vec{x} to \vec{y} , and for all t , the probability of reaching Γ within time t starting at \vec{y} is at least the probability of reaching Γ within time t starting at \vec{x} . Note that by this definition Γ must be absorbing. Intuitively, perturbation of propensities in monotonic systems only change how fast the system approaches the outcome. Indeed, we can bound the deviations in the outcome probability of any ρ -perturbation at any time by two specific ρ -perturbations, which are the maximally slowed down and sped up versions of the original process. This implies that monotonic SSA processes are robust at all times t when the outcome probability does not change quickly with t , and thus slowing down or speeding up the SSA process does not significantly affect the probability of the outcome.

For an SSA process or ρ -perturbation \mathcal{C} and set of states Γ , define $F^\Gamma(\mathcal{C}, t)$ to be the probability of being in Γ at time t . For SSA process \mathcal{C} , let $\tilde{\mathcal{C}}^{-\rho}$ be the ρ -perturbation defined by the constant deviations $\xi_j(t) = 1 - \rho$. Similarly, let $\tilde{\mathcal{C}}^{+\rho}$ be the ρ -perturbation defined by the constant deviations $\xi_j(t) = 1 + \rho$.

Lemma 7.8.4. *If an SSA process \mathcal{C} is monotonic for outcome Γ , then for any ρ -perturbation $\tilde{\mathcal{C}}$ of \mathcal{C} , $F^\Gamma(\tilde{\mathcal{C}}^{-\rho}, t) \leq F^\Gamma(\tilde{\mathcal{C}}, t) \leq F^\Gamma(\tilde{\mathcal{C}}^{+\rho}, t)$.*

Proof. If an SSA process is monotonic, allowing extra “spontaneous” transitions (as long as they are legal according to the SSA process) cannot induce a delay in entering Γ . We can decompose a perturbation with $\xi_j(t) \geq 1$ as the SSA process combined with some extra probability of reaction occurrence in the next interval dt . Thus, for a perturbation $\tilde{\mathcal{C}}$ of a monotonic SSA process \mathcal{C} in which $\xi_j(t) \geq 1$, we have $F^\Gamma(\mathcal{C}, t) \leq F^\Gamma(\tilde{\mathcal{C}}, t)$. By a similar argument, if $\tilde{\mathcal{C}}$ has $\xi_j(t) \leq 1$, then $F^\Gamma(\tilde{\mathcal{C}}, t) \leq F^\Gamma(\mathcal{C}, t)$. Now $\tilde{\mathcal{C}}^{-\rho}$ and $\tilde{\mathcal{C}}^{+\rho}$ are themselves monotonic SSA processes (\mathcal{C} scaled in time). Then by the above bounds, for any ρ -perturbation $\tilde{\mathcal{C}}$ of \mathcal{C} we have $F^\Gamma(\tilde{\mathcal{C}}^{-\rho}, t) \leq F^\Gamma(\tilde{\mathcal{C}}, t) \leq F^\Gamma(\tilde{\mathcal{C}}^{+\rho}, t)$. \square

Since $\tilde{\mathcal{C}}^{-\rho}$ and $\tilde{\mathcal{C}}^{+\rho}$ are simply the original SSA process \mathcal{C} scaled in time by a factor of $1/(1 + \rho)$ and $1/(1 - \rho)$, respectively, we can write the bound of the above lemma as $F^\Gamma(\mathcal{C}, t/(1 + \rho)) \leq F^\Gamma(\tilde{\mathcal{C}}, t) \leq F^\Gamma(\mathcal{C}, t/(1 - \rho))$. Rephrasing Lemma 7.8.4:

Corollary 7.8.1. *If an SSA process \mathcal{C} is monotonic for outcome Γ then it is (ρ, δ) -robust with respect to Γ at time t where $\delta = F^\Gamma(\tilde{\mathcal{C}}^{+\rho}, t) - F^\Gamma(\tilde{\mathcal{C}}^{-\rho}, t) = F^\Gamma(\mathcal{C}, t/(1 - \rho)) - F^\Gamma(\mathcal{C}, t/(1 + \rho))$.*

For many SSA processes, it may not be obvious whether they are monotonic. We would like a simple “syntactic” property of the SCRNs that guarantees monotonicity and can be easily checked. The following lemma makes it easy to prove monotonicity in some simple cases.

Lemma 7.8.5. *Let \mathcal{C} be an SSA process and Γ an outcome of SCRNs \mathcal{S} . If every species is a reactant in at most one reaction in \mathcal{S} , and there is a set $\{n_j\}$ such that outcome Γ occurs as soon as every reaction R_j has fired at least n_j times, then \mathcal{C} is monotonic with respect to Γ .*

Proof. The restriction on Γ allows us phrase everything in terms of counting reaction occurrences. For every reaction R_j , define $F_j(n, t)$ to be the probability that R_j has fired at least n times within time t . Now suppose we induce some reaction to fire by fiat. The only way this can decrease some $F_j(n, t)$ is if it decreases the count of a reactant of R_j or makes it more likely that some reaction $R_{j'}$ ($j' \neq j$) will decrease the count of a reactant of R_j . Either possibility is avoided if the SCRNs has the property that any species is a reactant in at most one reaction. Since $F^\Gamma(\mathcal{C}, t) = \prod_j F_j(n_j, t)$, this quantity cannot decrease as well, and monotonicity follows. \square

$O(1)\varepsilon^{O(1)n^{\uparrow\downarrow}+O(1)}$ term. Then the effect of a $\downarrow\downarrow$ leap would scale with ε , as does the effect of an \uparrow leap.

7.8.3 Robust Embedding of a TM in an SCRNs

Since we are trying to bound how the complexity of the prediction problem scales with increasing bounds on t and C but not with different SCRNs, we need a method of embedding a TM in an SCRNs in which the SCRNs is independent of the input length. Among such methods available ([4], Chapter 6), asymptotically the most efficient and therefore best for our purposes is the construction of Angluin et al. This result is stated in the language of distributed multi-agent systems rather than molecular systems; the system is a well-mixed set of “agents” that randomly collide and exchange information. Each agent has a finite state. Agents correspond to molecules (the system preserves a constant molecular count m); states of agents correspond to the species, and interactions between agents correspond to reactions in which both molecules are potentially transformed.

Now for the details of the SCRNs implementation of Angluin’s protocol. Suppose we construct an SCRNs corresponding to the Angluin et al. system as follows: Agent states correspond to species (i.e., for every agent state i there is a unique species S_i). For every pair of species S_{i_1}, S_{i_2} , ($i_1 \leq i_2$) we add reaction $S_{i_1} + S_{i_2} \rightarrow S_{i_3} + S_{i_4}$ if the population protocol transition function specifies $(i_1, i_2) \mapsto (i_3, i_4)$. Note that we allow null reactions of the form $S_{i_1} + S_{i_2} \rightarrow S_{i_1} + S_{i_2}$ including for $i_1 = i_2$. For every reaction R_j , we’ll use rate constant $k_j = 1$. The sum of all reaction propensities is $\lambda = \frac{m(m-1)}{2V}$ since every molecule can react with any other molecule.* The time until next reaction is an exponential random variable with rate λ . Note that the transition probabilities between SCRNs states are the same as the transition probabilities between the corresponding configurations in the population protocol since in the SCRNs every two molecules are equally likely to react next. Thus the SSA process is just a continuous time version of the population protocol process (where unit “time” expires between transitions). Therefore the SCRNs can simulate a TM in the same way as the population protocol.

But first we need to see how does time measured in the number of interactions correspond to the real-valued time in the language of SCRNs?

Lemma 7.8.6. *If the time between population protocol interactions is an exponential random variable with rate λ , then for any positive constants c, c_1, c_2 such that $c_1 < 1 < c_2$, there is N_0 such that for all $N > N_0$, N interactions occur between time $c_1 N/\lambda$ and $c_2 N/\lambda$ with probability at least $1 - N^{-c}$.*

Proof. The Chernoff bound for the left tail of a gamma random variable T with shape parameter N and rate λ is $\Pr[T \leq t] \leq (\frac{\lambda t}{N})^N e^{N-\lambda t}$ for $t < N/\lambda$. Thus $\Pr[T \leq c_1 N/\lambda] \leq (c_1 e^{1-c_1})^N$. Since $c_1 e^{1-c_1} < 1$ when $c_1 \neq 1$, $\Pr[T \leq c_1 N/\lambda] < N^{-c}$ for large enough N . An identical argument applies to the right tail Chernoff bound $\Pr[T \geq t] \leq (\frac{\lambda t}{N})^N e^{N-\lambda t}$ for $t > N/\lambda$. \square

The following lemma reiterates that an arbitrary computational problem can be embedded in a chemical system, and also shows that the chemical computation is robust with respect to the outcome of the computation. For a given TM and agent count m , let x_{f_0} and x_{f_1} be SCRNs states corresponding to the TM halting with a 0 and 1 output respectively.

Lemma 7.8.7. *Fix a perturbation bound $\rho > 0$, $\delta > 0$, and a randomized TM M with a Boolean output. There is an SCRNs implementing Angluin et al.’s population protocol, such that if $M(x)$ halts in no more than t_{tm} steps using no more than s_{tm} time, then starting with the encoding of x and using $m = O(1)2^{s_{tm}}$ molecules, at any time $t \geq t_{ssa} = O(1)V t_{tm} \log^4(m)/m$ the SSA process is in x_{f_b} with probability that is within δ of the probability that $M(x) = b$. Further, this SSA process is (ρ, δ) -robust with respect to states x_{f_0} and x_{f_1} at all times $t \geq t_{ssa}$.*

The first part of the lemma states that we can embed an arbitrary TM computation in an SCRNs, such that the TM computation is performed fast and correctly with arbitrarily high probability. The second part states that this method can be made arbitrarily robust to perturbations of reaction propensities. The first part follows directly from the results of [4], while the second part requires some additional arguments on our part.

If we only wanted to prove the first part, fix any randomized TM M with a Boolean output and any constant $\delta > 0$. There is a population protocol of Angluin et al. that can simulate the TM’s computation

*Just to confirm, splitting the reactions between the same species and between different species, the sum of the propensities is $\sum_i \frac{x_i(x_i-1)}{2V} + \sum_{i < i'} \frac{x_i x_{i'}}{V} = \frac{1}{2V} (\sum_i x_i x_i - \sum_i x_i + 2 \sum_{i < i'} x_i x_{i'}) = \frac{1}{2V} (\sum_{i, i'} x_i x_{i'} - \sum_i x_i) = \frac{n(n-1)}{2V}$ using the fact that $2 \sum_{i < i'} x_i x_{i'} = \sum_{i \neq i'} x_i x_{i'}$ and $\sum_i x_i x_i + \sum_{i \neq i'} x_i x_{i'} = \sum_{i, i'} x_i x_{i'}$.

on arbitrary inputs as follows: If on some input x , M uses t_{tm} computational time and s_{tm} space, then the protocol uses $m = O(1)2^{s_{tm}}$ agents, and the probability that the simulation is incorrect or takes longer than $N = O(1)t_{tm}m \log^4 m$ interactions is at most $\delta/2$. This is proved by using Theorem 11 of [4], combined with the standard way of simulating a TM by a register machine using multiplication by a constant and division by a constant with remainder. The total probability of the computation being incorrect or lasting more than N interactions obtained is at most $O(1)t_{tm}m^{-c}$. Since for any algorithm terminating in t_{tm} steps, $2^{s_{tm}} \geq O(1)t_{tm}$, we can make sure this probability is at most $\delta/2$ by using a large enough constant in $m = O(1)2^{s_{tm}}$. By Lemma 7.8.6, the probability that $O(1)N$ interactions take longer than $O(1)N/\lambda$ time to occur is at most $\delta/2$. Thus the total probability of incorrectly simulating M on x or taking longer than $O(1)N/\lambda = O(1)Vt_{tm} \log^4(m)/m$ time is at most δ . The Boolean output of M is indicated by whether we end up in state $x_{f_0}^{\vec{1}}$ or $x_{f_0}^{\vec{0}}$. (If the computation was incorrect or took too long we can be in neither.) This proves the first part of the lemma.

We now sketch out the proof of how the robustness of the Angluin et al. system can be established, completing the proof of Lemma 7.8.7. The whole proof requires retracing the argument in the original paper; here, we just outline how this retracing can be done. First, we convert the key lemmas of their paper to use real-valued SCRNs time rather than the number of interactions. The consequences of the lemmas (e.g., that something happens before something else) are preserved and thus the lemmas can be still be used as in the original paper to prove the corresponding result for SCRNs. The monotonicity of the processes analyzed in the key lemmas can be used to argue that the overall construction is robust.

We need the following consequence of Lemma 7.8.4:

Corollary 7.8.2. *If an SSA process \mathcal{C} is monotonic for outcome Γ , and with probability p it enters Γ after time t_1 but before time t_2 , then for any ρ -perturbation $\tilde{\mathcal{C}}$ of \mathcal{C} , the probability of entering Γ after time $t_1/(1 + \rho)$ but before time $t_2/(1 - \rho)$ is at least p .*

Proof. Let $p_1 = F^\Gamma(\mathcal{C}, t_1)$ and $p_2 = F^\Gamma(\mathcal{C}, t_2)$. Using Lemma 7.8.4 we know that $\forall t, F^\Gamma(\mathcal{C}, t/(1 - \rho)) \geq F^\Gamma(\tilde{\mathcal{C}}, t)$. Thus, $p_1 = F^\Gamma(\mathcal{C}, t_1) \geq F^\Gamma(\tilde{\mathcal{C}}, (1 - \rho)t_1)$. Similarly we obtain $p_2 = F^\Gamma(\mathcal{C}, t_2) \leq F^\Gamma(\tilde{\mathcal{C}}, (1 + \rho)t_2)$. Thus $F^\Gamma(\tilde{\mathcal{C}}, (1 + \rho)t_2) - F^\Gamma(\tilde{\mathcal{C}}, (1 - \rho)t_1) \geq p_2 - p_1 = p$. \square

As an example let us illustrate the conversion of Lemma 2 of [4]. The lemma bounds the number of interactions to infect k agents in a ‘‘one-way epidemic’’ starting with a single infected agent. In the one-way epidemic, a non-infected agent becomes infected when it interacts with a previously infected agent. With our notation, this lemma states:

Let $N(k)$ be the number of interactions before a one-way epidemic starting with a single infected agent infects k agents. Then for any fixed $\varepsilon > 0$ and $c > 0$, there exist positive constants c_1 and c_2 such that for sufficiently large total agent count m and any $k > m^\varepsilon$, $c_1 m \ln k \leq N(k) \leq c_2 m \ln k$ with probability at least $1 - m^{-c}$.

For any m and k we consider the corresponding SSA process \mathcal{C} and outcome Γ in which at least k agents are infected. Since the bounds on $N(k)$ scale at least linearly with m , we can use Lemma 7.8.6 to obtain:

Let $t(k)$ be the time before a one-way epidemic starting with a single infected agent infects k agents. Then for any fixed $\varepsilon > 0$ and $c > 0$, there exist positive constants c_1 and c_2 such that for sufficiently large total agent count m and any $k > m^\varepsilon$, $c_1 m \ln(k)/\lambda \leq t(k) \leq c_2 m \ln(k)/\lambda$ with probability at least $1 - m^{-c}$.

Finally consider the SSA process of the one-way epidemic spreading. The possible reactions either do nothing (reactants are either both infected or both non-infected), or a new agent becomes infected. It is clear that for any m and k , \mathcal{C} is monotonic with respect to outcome Γ in which at least k agents are infected. This allows us to use Corollary 7.8.2 to obtain:

Fix any $\rho > 0$, and let $t(k)$ be the time before a one-way epidemic starting with a single infected agent infects k agents in some corresponding ρ -perturbation. Then for any fixed $\varepsilon > 0$, $c > 0$, there exist positive constants c_1 and c_2 such that for sufficiently large total agent count m and any $k > m^\varepsilon$, $c_1 m \ln(k)/(\lambda(1 + \rho)) \leq t(k) \leq c_2 m \ln(k)/(\lambda(1 - \rho))$ with probability at least $1 - m^{-c}$.

Since ρ is a constant, what we have effectively done is convert the result in terms of interactions to a result in terms of real-valued time that is robust to ρ -perturbations simply by dividing by λ and using different multiplicative constants.

The same process can be followed for the key lemmas of Angluin et al. (Lemma 3 through Lemma 8). This allows us to prove a robust version of Theorem 11 of Angluin et al. by retracing the argument of their paper using the converted lemmas and the real-valued notion of time throughout. Since the only way that time is used is to argue that something occurs before something else, the new notion of time, obtained by dividing by λ with different constants, can always be used in place of the number of interactions.

7.8.4 Proof of Theorem 7.6.1: Lower Bound on the Computational Complexity of the Prediction Problem

In this section we prove Theorem 7.6.1 from the text which lower-bounds the computational complexity of the prediction problem as a function of m , t , and C . The bound holds even for arbitrarily robust SSA processes. The theorem shows that this computational complexity is at least linear in t and C , as long as the dependence on m is at most polylogarithmic. The result is a consequence of the robust embedding of a TM in an SCRN (Lemma 7.8.7).

Let the prediction problem be specified by giving the SSA process (via the initial state and volume), the target time t , and the target outcome Γ in some standard encoding such that whether a state belongs to Γ can be computed in time polylogarithmic in m .

Theorem. *Fix any perturbation bound $\rho > 0$ and $\delta > 0$. Assuming the hierarchy conjecture (Conjecture 7.6.1), there is an SCRN \mathcal{S} such that for any prediction algorithm \mathcal{A} and constants $c_1, c_2, \beta, \eta, \gamma > 0$, there is an SSA process \mathcal{C} of \mathcal{S} and a $(m, t, C, 1/3)$ -prediction problem \mathcal{P} of \mathcal{C} such that \mathcal{C} is (ρ, δ) -robust with respect to \mathcal{P} and \mathcal{A} cannot solve \mathcal{P} in computational time $c_1 (\log m)^\beta t^\eta (C + c_2)^\gamma$ if $\eta < 1$ or $\gamma < 1$.*

Suppose someone claims that for any fixed SCRN, they can produce an algorithm for solving $(m, t, C, 1/3)$ -prediction problems for SSA processes of this SCRN assuming the SSA process is (ρ, δ) -robust with respect to the prediction problem for some fixed ρ and δ , and further they claim the algorithm runs in computation time at most

$$O(1) (\log(m))^\beta t^\eta (C + O(1))^\gamma \quad (7.11)$$

for some $\eta < 1$ ($\beta, \gamma > 0$). We argue that assuming the hierarchy conjecture is true, such a value of η is impossible.

To achieve a contradiction of the hierarchy conjecture, consider any function probabilistically computable in $t_{tm}(n) = O(1)n^\zeta$ time and $s_{tm}(n) = O(1)n$ space for $\zeta = \frac{\beta+4\eta}{1-\eta} + 1$. Construct a randomized TM having error at most $1/24$ by running the original randomized TM $O(1)$ times and taking the majority vote. Use Lemma 7.8.7 to encode the TM probabilistically computing this function in a (ρ, δ) -robust SSA process such that the error of the TM simulation is at most $1/24$. Then predicting whether the process ends up in state $x_{f_0}^{\vec{f}}$ or $x_{f_1}^{\vec{f}}$ provides a probabilistic algorithm for computing this function. The resulting error is at most $1/24 + 1/24 + 1/3 = 5/12 < 1/2$, where the first term $1/24$ is the error of the TM, the second term $1/24$ is for the additional error of the TM embedding in the SSA process, and the last term $1/3$ is for the allowed error of the prediction problem. By repeating $O(1)$ times and taking the majority vote, this error can be reduced below $1/3$, thereby satisfying the definition of probabilistic computation. How long does this method take to evaluate the function? We use $V = m$ so that C is a constant, resulting in $t_{ssa} = O(1)t_{tm}(n) \log^4 m = O(1)n^{\zeta+4}$ since $m = O(1)2^n$. Setting up the prediction problem by specifying the SSA process (via the initial state and volume), target final state and time t_{ssa} requires $O(1) \log m = O(1) n$ time.* Then the prediction problem is solved in computation time $O(1)(\log(m))^\beta t_{ssa}^\eta = O(1)n^{\beta+(\zeta+4)\eta}$. Thus the total computation time is $O(1)(n^{\beta+(\zeta+4)\eta} + n)$ which, by our choice of ζ , is less than $O(1)n^\zeta$, leading to a contradiction of the hierarchy conjecture.

*By the construction of [4], setting up the initial state requires letting the binary expansion of the molecular count of a certain species be equal the input. Since the input is given in binary and all molecular counts are represented in binary, this is a linear time operation. Setting up the final state $x_{f_0}^{\vec{f}}$ or $x_{f_1}^{\vec{f}}$ is also linear time. Computing the target time for the prediction problem t_{ssa} is asymptotically negligible.

Is $\gamma < 1$ possible? Observe that if $\gamma < \eta$ then the claimed running time of the algorithm solving the prediction problem (expression 7.11) with time $t_{ssa} = O(1)Vt_{tm}(n)\log^4(m)/m$ can be made arbitrarily small by decreasing V . This leads to contradiction of the hierarchy conjecture. Therefore $\gamma \geq \eta \geq 1$.

7.8.5 On Implementing BTL on a Randomized TM

The idealized BTL algorithm presented in Section 7.5.1 relies on infinite precision real-value arithmetic, while only finite precision floating-point arithmetic is possible on a TM. Further, the basic randomness generating operation available to a randomized TM is choosing one of a fixed number of alternatives uniformly, which forces gamma and binomial draws to be approximated. This complicates estimates of the computation time required per leap, and also requires us to ensure that we can ignore round-off errors in floating-point operations and tolerate approximate sampling in random number draws.

Can we implement gamma and binomial random number generators on a randomized TM and how much computational time do they require? It is easy to see that arbitrary precision uniform $[0, 1]$ random variates can be drawn on a randomized TM in time linear in precision. It is likely that approximate gamma and binomial random variables can be drawn using methods available in the numerical algorithms literature which uses uniform variate draws as the essential primitive. Since many existing methods for efficiently drawing (approximate) gamma and binomial random variables involve the rejection method, the computation time for these operations is likely to be an expectation. Specifically, it seems reasonable that drawing gamma and binomial random variables can be approximately implemented on a randomized TM such that the expected time of these operations is polynomial in the length of the floating-point representation of the distribution parameters and the resultant random quantity.*

The computational complexity of manipulating integer molecular counts on a TM is polylogarithmic in m . Let l be an upper bound on the expected computational time required for drawing the random variables and real number arithmetic; l is potentially a function of m , V , t , and the bits of precision used. Using Markov's inequality and Theorem 7.5.1 we can then obtain a bound on the total computation time that is true with arbitrarily high probability. We also make the TM keep track of the total number of computational steps it has taken[†] and cut off computation when it exceeds the expectation by some fixed factor. Then we obtain the following bound on the total computation time: $O(1)((\log(m))^{O(1)} + l)t(C + O(1))$.

We have three sources of error. First, since BTL simulates a ρ -perturbation rather than the original SSA process, the probability of the outcome may be off by δ_1 , assuming the SSA process was (ρ, δ_1) -robust. Further, since we are using finite precision arithmetic and only approximate random number generation, the deviation from the correct probability of the outcome may increase by another δ_2 . Finally, there is a δ_3 probability that the algorithm cuts off computation before it completes. We have assumed a fixed $\delta_1 < \delta$, where δ is the allowed error of the prediction problem. We can make δ_3 an arbitrarily small constant by increasing the total computation time by a constant factor (using Markov's inequality). Further, let us assume that δ_2 is small enough to ensure that the total error $\delta_1 + \delta_2 + \delta_3 \leq \delta$ fulfills the requirements of solving the (m, t, C, δ) -prediction problem.[‡]

Bibliography

- [1] D. Adalsteinsson, D. McMillen, and T. C. Elston. Biochemical network stochastic simulator (BioNetS): software for stochastic modeling of biochemical networks. *BMC Bioinformatics*, 5, 2004.

*The numerical algorithms literature, which assumes that basic floating point operations take unit time, describes a number of algorithms for drawing from an (approximate) standard gamma distribution [2], and from a binomial distribution [20], such that the expected number of floating-point operations does not grow as a function of distribution parameters (however, some restrictions on the parameters may be required). On a TM basic arithmetic operations take polynomial time in the length of the starting numerical values and the calculated result.

[†]Compute the bound and write this many 1s on a work tape, and after each computational step, count off one of the 1s until no more are left.

[‡]We conjecture that for any fixed δ_2 , we can find some fixed amount of numerical precision to not exceed δ_2 for (ρ, δ_1) -robust processes. We would like to show that robustness according to our definition implies robustness to round-off errors and approximate random number generation. While this conjecture has strong intuitive appeal, it seems difficult to prove formally, and represents an area for further study.

- [2] J. Ahrens and U. Dieter. Generating Gamma Variates by a Modified Rejection Technique. *Language*, 54:853–882, 1978.
- [3] U. Alon. *An Introduction to Systems Biology: Design Principles of Biological Circuits*. Chapman & Hall/CRC, 2007.
- [4] D. Angluin, J. Aspnes, and D. Eisenstat. Fast computation by population protocols with a leader. Technical Report YALEU/DCS/TR-1358, Yale University Department of Computer Science, 2006. Extended abstract to appear, DISC 2006.
- [5] A. P. Arkin, J. Ross, and H. H. McAdams. Stochastic kinetic analysis of a developmental pathway bifurcation in phage- λ Escherichia coli. *Genetics*, 149:1633–1648, 1998.
- [6] B. Barak. A probabilistic-time hierarchy theorem for ‘slightly non-uniform’ algorithms. In *Proceedings of International Workshop on Randomization and Computation*, 2002.
- [7] Y. Cao, D. Gillespie, and L. Petzold. Efficient step size selection for the tau-leaping simulation method. *The Journal of Chemical Physics*, 124:044109, 2006.
- [8] M. B. Elowitz, A. J. Levine, E. D. Siggia, and P. S. Swain. Stochastic gene expression in a single cell. *Science*, 297:1183–1185, 2002.
- [9] P. Érdi and J. Tóth. *Mathematical Models of Chemical Reactions: Theory and Applications of Deterministic and Stochastic Models*. Manchester University Press, 1989.
- [10] S. N. Ethier and T. G. Kurtz. *Markov Processes: Characterization and Convergence*. John Wiley & Sons, 1986.
- [11] L. Fortnow and R. Santhanam. Recent work on hierarchies for semantic classes. *SIGACT News*, 37:36–54, 2006.
- [12] M. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *Journal of Physical Chemistry A*, 104:1876–1889, 2000.
- [13] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81:2340–2361, 1977.
- [14] D. T. Gillespie. A rigorous derivation of the chemical master equation. *Physica A*, 188:404–425, 1992.
- [15] D. T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *The Journal of Chemical Physics*, 115, 2001.
- [16] D. T. Gillespie. Improved leap-size selection for accelerated stochastic simulation. *The Journal of Chemical Physics*, 119(16):8229, 2003.
- [17] D. T. Gillespie. Stochastic simulation of chemical kinetics. *Annual Review of Physical Chemistry*, 58:35–55, 2007.
- [18] P. Guptasarma. Does replication-induced transcription regulate synthesis of the myriad low copy number proteins of Escherichia coli? *Bioessays*, 17:987–997, 1995.
- [19] F. Horn and R. Jackson. General mass action kinetics. *Archive for Rational Mechanics and Analysis*, 47(2):81–116, 1972.
- [20] V. Kachitvichyanukul and B. Schmeiser. Binomial random variate generation. *Communications of the ACM*, 31(2):216–222, 1988.
- [21] A. M. Kierzek. STOCKS: STOChastic kinetic simulations of biochemical systems with Gillespie algorithm. *Bioinformatics*, 18:470–481, 2002.

- [22] T. G. Kurtz. The relationship between stochastic and deterministic models for chemical reactions. *The Journal of Chemical Physics*, 57:2976–2978, 1972.
- [23] B. Levin. *Genes VII*. Oxford University Press, 1999.
- [24] M. Malek-Mansour and G. Nicolis. A master equation description of local fluctuations. *Journal of Statistical Physics*, 13:197–217, 1975.
- [25] H. H. McAdams and A. P. Arkin. Stochastic mechanisms in gene expression. *Proceedings of the National Academy of Sciences*, 94:814–819, 1997.
- [26] D. A. McQuarrie. Stochastic approach to chemical kinetics. *Journal of Applied Probability*, 4:413–478, 1967.
- [27] M. Morohashi, A. E. Winn, M. T. Borisuk, H. Bolouri, J. Doyle, and H. Kitano. Robustness as a Measure of Plausibility in Models of Biochemical Networks. *Journal of Theoretical Biology*, 216(1):19–30, 2002.
- [28] M. Rathinam and H. El Samad. Reversible-equivalent-monomolecular tau: A leaping method for “small number and stiff” stochastic chemical systems. *Journal of Computational Physics*, 224(2):897–923, 2007.
- [29] M. Rathinam, L. Petzold, Y. Cao, and D. Gillespie. Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method. *The Journal of Chemical Physics*, 119:12784, 2003.
- [30] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing, 1997.
- [31] E. Sontag. Monotone and Near-Monotone Systems. *Lecture Notes in Control and Information Sciences*, 357:79, 2007.
- [32] G. M. Suel, J. Garcia-Ojalvo, L. M. Liberman, and M. B. Elowitz. An excitable gene regulatory circuit induces transient cellular differentiation. *Nature*, 440:545–550, 2006.
- [33] N. van Kampen. *Stochastic Processes in Physics and Chemistry*. Elsevier, revised edition, 1997.
- [34] K. Vasudeva and U. S. Bhalla. Adaptive stochastic-deterministic chemical kinetic simulations. *Bioinformatics*, 20:78–84, 2004.

Chapter 8

Enzyme-Free Nucleic Acid Logic Circuits

Collaborators: Georg Seelig, David Yu Zhang, and Erik Winfree. **My contribution:** GS and I came up with the basic idea, informed by some constructions of DYZ. I performed a couple of the experiments, but the majority was done by GS, including data analysis. The text was written by all.

This chapter was published as: Georg Seelig, David Soloveichik, David Yu Zhang, Erik Winfree, “Enzyme-Free Nucleic Acid Logic Circuits,” *Science*, 314:1585-1588, 2006. (Supplementary information appears online.)

8.1 Abstract

Biological organisms perform complex information processing and control tasks using sophisticated biochemical circuits, yet the engineering of such circuits remains ineffective compared to that of electronic circuits. To systematically create complex yet reliable circuits, electrical engineers employ digital logic wherein gates and subcircuits are composed modularly and signal restoration prevents signal degradation. We report the design and experimental implementation of DNA-based digital logic circuits. We demonstrate AND, OR, and NOT gates, signal restoration, amplification, feedback, and cascading. Gate design and circuit construction is modular. The gates use single-stranded nucleic acids as inputs and outputs, and the mechanism relies exclusively on sequence recognition and strand displacement. Biological nucleic acids such as microRNAs can serve as inputs, suggesting applications in biotechnology and bioengineering.

8.2 Introduction

To date, no man-made chemical circuits even remotely approach the complexity and reliability of silicon-based electronics. Once reliable principles for their design are established, synthetic chemical circuits could be used routinely to control nanoscale devices *in vitro*, to analyze complex chemical samples *in situ*, or to interface with existing biological circuits *in vivo* [7]. Construction of synthetic biological circuits *de novo* is a powerful test of design principles [15].

Rational design of nucleic acid devices is simplified by the predictability of Watson-Crick base pairing; thus nucleic acids are a promising alternative to proteins for synthetic chemical circuits. Allosteric ribozymes that take small molecules as input have been shown to perform logical functions [4]; however, their output (a cleaved or ligated oligonucleotide) is of a different form than the input, hence cascading is difficult. Automata performing multiple logical operations in parallel [17], single-step signaling cascades [12], and a feedback cycle that acts as an exponential chain reaction [10] were built using deoxyribozymes controlled by input oligonucleotides [16]. Another approach utilizes sequence recognition to control enzyme catalysis of covalent bond formation and breakage [21, 3, 2]. Alternatively, nucleic-acid reactions can be driven without enzyme or (deoxy)ribozyme catalysis [22, 18]; this principle has been exploited to construct DNA-based

logic gates and signaling cascades [5, 6]. Such molecular automata may give rise to “smart” therapeutics for medical applications [21, 16, 2]. Recently, engineered nucleic-acid logic switches based on hybridization and conformational changes have been successfully demonstrated in vivo [8, 1]. The remaining challenge is to design chemical logic gates that can be combined to construct large, reliable circuits. The analogous challenge for engineering electronic circuits was met by the development of digital design principles;* likewise these may prove essential for designing complex, yet robust, chemical circuits.

We report the construction of in vitro DNA-based logic gates and circuits that embody digital design principles: logic, cascading, restoration, fan-out, and modularity. These circuits implement a complete set of Boolean logic functions (AND, OR, and NOT) using short oligonucleotides as input and output. Because the input and output are of the same form, the gates can be cascaded to create multilayer circuits. Logical values “0” and “1” are represented by low and high concentrations, respectively. Signal restoration is performed by threshold and amplifier gates that protect against noise, signal loss, and leaky reactions. Amplifier gates can also be used to ensure that a logic gate generates sufficient signal to drive multiple downstream targets. Watson-Crick interactions between modular recognition domains determine the connectivity of gates. Sequences can be chosen with few constraints, allowing the construction of arbitrary circuits with negligible cross-activation. Furthermore, modular construction allows for interfacing with existing molecular components — be they pre-designed subcircuits or naturally occurring nucleic acids.

8.3 Gate Construction and Verification

Gate function is entirely determined by base pairing and breaking. Every gate consists of one or more gate strands and one output strand (Figs. 8.1A and S1). The output strand either serves as an input to a downstream gate or it is modified with a dye-label to provide a readout in a fluorescence experiment. Both ends of the output strand (Fig. 8.1A), or only one end (translator gates in Fig. 8.2), can be attached to the gate complex. Fig. 8.1A shows an AND-gate assembled from an output strand and two gate strands. Addition of single-stranded inputs to a solution containing the gate initiates a computation. Each gate strand contains a recognition region that is complementary to its input. Initially the recognition regions of all gate strands are double-stranded and therefore inert, except for the toehold farthest from the output strand (strand G in Fig. 8.1A). When the first input binds this toehold, it displaces the first gate strand by three-way branch migration [9, 20], exposing the toehold for the subsequent input and releasing an inert double-stranded waste product. A similar process can now occur for the second input. The output strand is released if and only if both inputs are present. To implement this design, DNA sequences (Tables S1–S3) were selected to ensure correct complementarity while minimizing spurious interactions [23].

The two-input AND gate has four entries in its truth table (Fig. 8.1B) and has been shown to function correctly, using fluorescence kinetics experiments and gel electrophoresis (Figs. 8.1C–D). Multi-input AND gates can also be designed using the same principles and have been shown to work reliably (Fig. S2). The gates in all our experiments were purified by gel electrophoresis after triggering “leaky” complexes ([23], Fig. S3).

8.4 Circuit Construction

The output strand of one gate may be an input strand to a downstream gate. It is essential that the output strand does not interact with downstream gates prior to release. Protecting the toehold binding region of output strands in upstream gates prevents such interactions. We built a circuit composed of one AND gate and two translator gates that demonstrates this principle (Fig. 8.2A and S4). A translator gate converts the signal encoded in the input strand to the signal encoded in the output strand and is implemented as a single input AND gate. The translator gates JK and LM translate two biological microRNA sequences (mouse let-7c and mir-124) into outputs with recognition regions identical to strands G_{in} and F_{in} . The input to a

*In contrast to digital electronic circuits, analog electronic circuits have not advanced rapidly because circuit design remains more “art” than systematic engineering, making the construction of large reliable circuits difficult. This is often attributed to the lack of the digital abstraction: in analog circuits even slight signal changes carry meaning (e.g., the value is 5.2 not 5.3) and thus restoration to clean up noise or gate malfunction is not possible. The lack of restoration also complicates circuit modularity, because circuit behavior can be subtly changed when subcircuits are combined. See, e.g., [13].

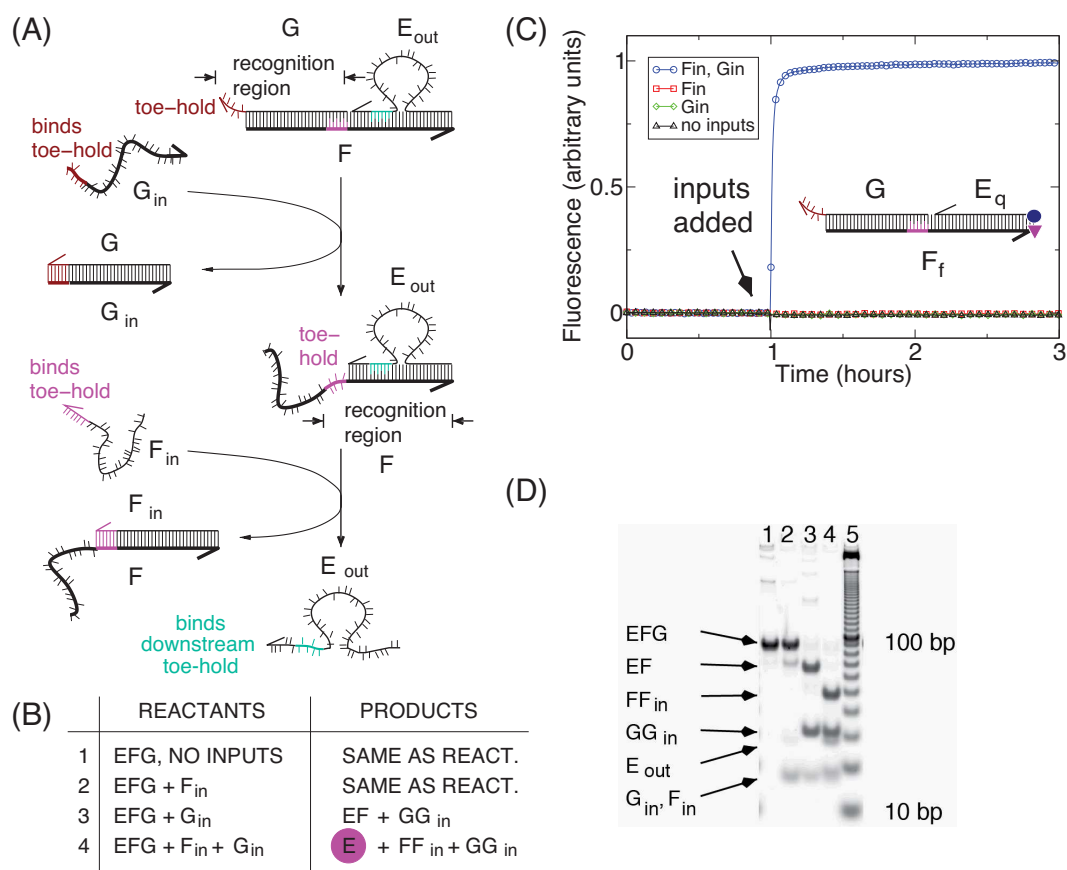


Figure 8.1: Two-input AND gate. (A) The gate consists of three DNA strands, E_{out} (57 mer), F (60 mer), and G (36 mer). The 3'-ends are marked by arrows. Toeholds and toehold binding regions (all 6 nt) are indicated in color. Input strands F_{in} and G_{in} (36 mers) are complementary to recognition regions within the corresponding gate strands F and G . (B) Truth table for the two-input AND gate. The released output strand is highlighted. (C) In fluorescence experiments, strands F_f (TAMRA fluorophore at the 3'-end) and E_q (Iowa Black RQ quencher at the 5'-end, without bulge loop) were used instead of F and E_{out} (see inset). Release of output strand results in increased fluorescence. Experiments conducted at 25° C with gate concentrations of 250 nM and input concentrations of 300 nM in a Tris-acetate-EDTA buffer containing 12.5 mM Mg^{++} . (D) Non-denaturing gel electrophoresis directly confirms reaction intermediates and waste products for each possible input combination. Lanes 1–4: The samples are as described in entries 1–4 of the truth table. The gate used in this experiment is as shown in (A). Lane 5: 10 bp ladder.

translator gate and the recognition region of its output strand need only share sequence in the toehold region. If two translators are cascaded then there is no sequence restriction between the initial input strand and the final output strand. This is called a full translator; the cascading of *NO* and *HI* is an example (Fig. 8.3 and Fig. S1). Translators can connect subcircuits that do not a priori use the same sequences for the toehold and recognition regions. This is particularly useful for adapting an existing circuit to compute on arbitrary biological inputs.

The circuit of Fig. 8.2A was also tested under conditions relevant to potential biological applications. The circuit works comparably with RNA inputs as with DNA inputs because gate function depends solely on Watson-Crick complementarity (Fig. 8.2A and Fig. S4). Increasing the temperature to 37 °C does not degrade circuit performance. Finally, the circuit functions well in the presence of potentially interfering biological RNA (mouse brain total RNA) at a concentration in excess of gate complexes and input strands.

Since a small set of logic gates (AND, OR, and NOT) is sufficient for effective computation of any

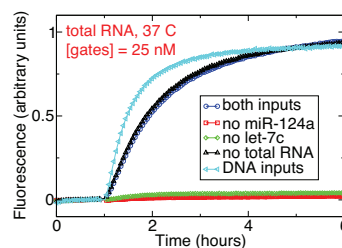
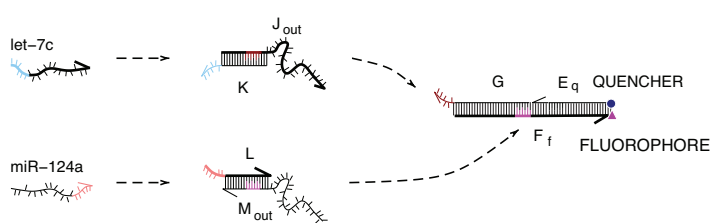
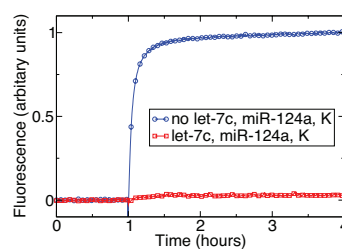
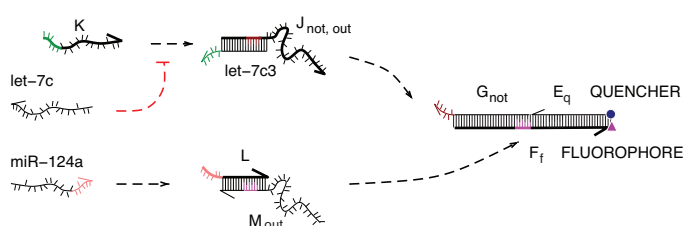
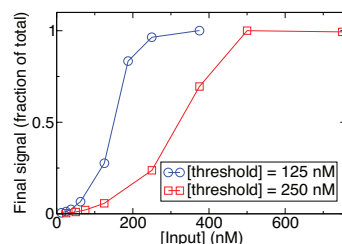
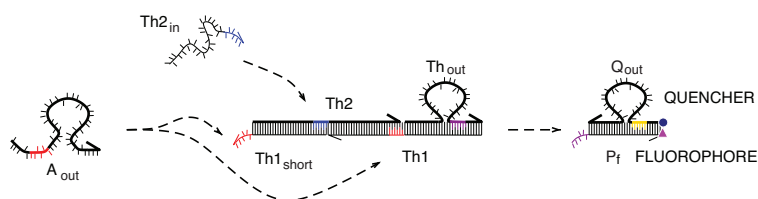
A CIRCUIT DIAGRAM FOR: let-7c AND miR-124a**B** CIRCUIT DIAGRAM FOR: (NOT let-7c) AND miR-124a**C** CIRCUIT DIAGRAM FOR THRESHOLDING

Figure 8.2: Translator gates, NOT operation and signal restoration. Dashed arrows indicate where input or output strands can serve as inputs to downstream gates. (A) Circuit operation at 37 °C with RNA inputs and DNA gates in a total RNA background. All gates are at 25 nM, synthetic RNA inputs are at 30 nM, and total RNA (mouse brain) is at a concentration of 200 $\mu\text{g}/\text{ml}$. Proper function is observed. For comparison, experiments with no total RNA were performed, using either both RNA inputs or both DNA inputs. (B) The NOT gate consists of a translator gate and an inverter strand complementary to *let-7c*. Gate, inverter strand, and input concentrations are 250 nM, 300 nM, and 300 nM, respectively. Here and in all following experiments the temperature was 25 °C and DNA equivalents of the biological microRNAs were used. If *let-7c* is present, inverter strand *K* will preferentially hybridize to *let-7c*. Otherwise, inverter strand *K* will trigger the translator. (C) The thresholding gate, using a dye/quencher-labeled “read-out” gate to monitor the output. Strand *Th2_{in}* is part of the thresholding unit and is added before the start of the experiment. The final fluorescence is plotted against the input concentration for two different concentrations of the threshold gate.

Boolean function, we developed DNA gates to perform these operations. Logical OR functionality is obtained by using two gates that produce the same output. We constructed a three-gate chemical circuit in which a logical OR feeds into a logical AND (Fig. S4B). Acting as a logical OR, translator gates *ST* and *UV* take different inputs (*mir-15a* and *mir-10b*) but release outputs with identical recognition regions. If Boolean values are represented by the presence of either one strand (0) or another strand (1) — the so-called “dual-rail” representation [11] — then AND and OR are themselves sufficient to compute any Boolean function.

If a Boolean value is represented by the presence or absence of a single input strand, a NOT gate may be necessary. We modified the circuit shown in Fig. 8.2A to invert the *let-7c* input (Fig. 8.2C). The NOT gate makes use of an additional “inverter” strand that triggers the gate unless the input strand is present to act as a competitive inhibitor. Since the inverter strand must be added simultaneously with the input, NOT gates are restricted to the first layer of the circuit. This is sufficient to create a dual-rail representation from which arbitrary subsequent computation can be performed with just AND and OR.

CIRCUIT DIAGRAM FOR: let-7c AND miR-124a AND (miR-15a OR miR-10b) AND (miR-143 OR miR-122a)

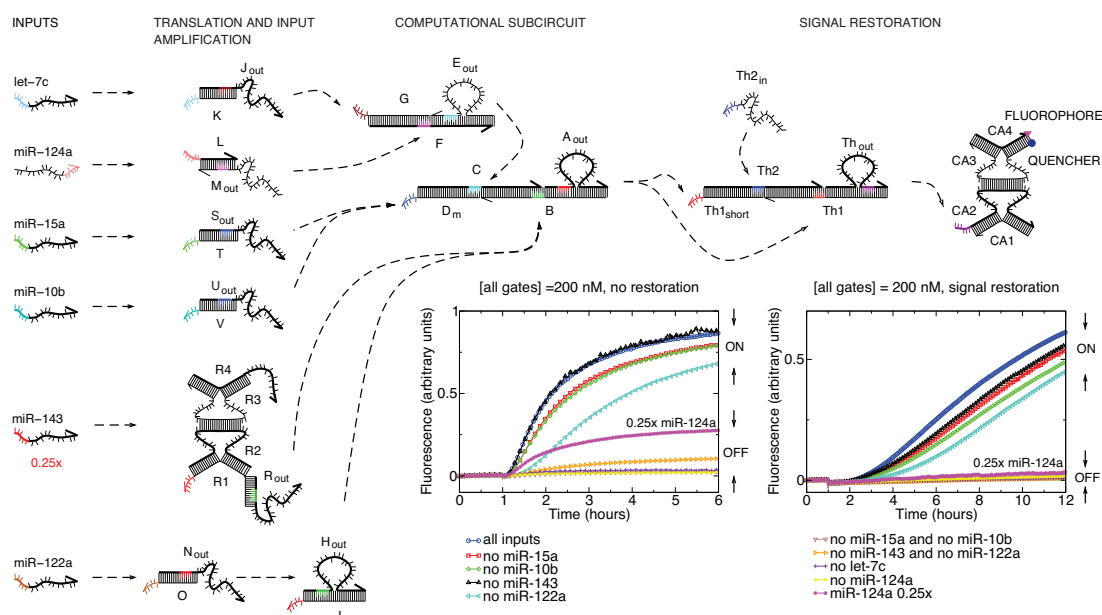


Figure 8.3: Signal propagation through a complex chemical circuit combining AND, OR, sequence translation, input amplification, and signal restoration. The 5-layer circuit consists of a total of eleven gates and accepts six inputs. With the exception of the threshold gate which is at 100 nM ($Th2_{in}$ is at 150 nM), all gates are at 200 nM (1x) per gate. Unless otherwise specified, inputs were added at 250 nM (1.25x). Mir-143 was added at 50 nM (0.25x) and subsequently amplified by the input amplifier. Inset: fluorescence traces of circuit operation without and with the signal restoration module (threshold plus amplifier). The traces for input conditions corresponding to a logical TRUE output (“ON”) are clearly distinguishable from the logical FALSE output (“OFF”). Cases tested include when all inputs are present, all cases where exactly one input is missing, and combinations of inputs that turn off an OR clause. Assuming monotonicity, withholding additional inputs will never lead to a logical TRUE output. To determine the response of the circuit to a leaky OFF signal, input mir-124 was added at 50 nM (0.25x) while all other inputs were added normally.

A gate may fail in two ways: it may fail to produce enough output when triggered, or it may “leak” by spontaneously releasing the output strand. Both types of error require signal restoration; the former requires increasing a moderate output amount to the full activation level, and the latter requires decreasing a small output amount to a negligible level. To implement signal restoration, we developed gates for amplification and thresholding. The threshold gate (Fig. 8.2D) is a three-input AND gate with identical first and third inputs. The second input is only necessary for structural purposes; it is always present and can be considered part of the thresholding unit. A substoichiometric amount of input (with respect to threshold gates) will cause most gates to lose only their first and second gate strands — thus releasing no output. Input concentrations two-fold greater than the concentration of threshold gates will cause most gates to produce output. The threshold gate’s concentration sets the threshold for a sigmoidal non-linearity (Figs. 8.2D, S5, [23]).

Since the threshold gate’s output cannot exceed half the input signal, subsequent amplification is necessary. A hybridization-based system for catalytic amplification was demonstrated previously [14]. With minor modifications, the system serves as both an input amplifier and full translator (Fig. S6 and Fig. 8.3, left, mir-143 translator), or as a fluorescence readout (Figs. S7A and 8.3, right). Alternatively, amplifiers based on feedback logic can be designed (Fig. S6B). A threshold gate together with an amplifier gate constitutes a signal restoration module whose incorporation into large circuits at multiple intermediate points ensures the stability of digital representation [19].

Finally, to demonstrate modularity and scalability we composed eleven gates into a larger circuit. The circuit combines previously introduced modules for input translation and amplification, calculation of AND

and OR, and signal restoration (Fig. 8.3). The inputs to the circuit are DNA analogs of six mouse microRNAs. To determine the effectiveness of signal restoration, we constructed an equivalent circuit without signal restoration and tested both circuits with an input at 0.25x to simulate a large upstream leak. The complete circuit maintained a low output signal, whereas the circuit without signal restoration exhibited a $\approx 25\%$ output leak (Fig. 8.3, inset). To verify other circuit components, several subcircuits were constructed and tested independently (Figs. S8 and S9). The feedback fluorescence amplifier was tested as a replacement for the catalytic amplifier at the output, resulting in a circuit containing 12 gates (Fig. S10).

8.5 Conclusion

As increasingly larger circuits are constructed, speed becomes a limiting factor. The circuit without signal restoration takes 2 hours to reach half-activation (Fig. 8.3, inset, left). The circuit with signal restoration has two additional layers and takes 10 hours to achieve half-activation (Fig. 8.3, inset, right). Despite the slow operation, in both cases a clear difference between off and on states can be distinguished much earlier. Speeding up the responses of individual gates (e.g., by shortening recognition domains) or changing other reaction conditions may improve overall circuit performance.

Our success in creating large circuits can be attributed to: adherence to the tenets of digital logic, toehold sequestering combined with branch migration and strand displacement, reduction of leak reactions by purification, and modularity of design. The logic gates developed here and the principles they are based on can also be used to construct analog or hybrid circuits [13] and are likely to prove compatible with other approaches to building molecular automata in vitro and in vivo [16, 6, 12, 3, 2, 8, 1]. Since evidence suggests that our logic gates can use natural RNA as input, and that they behave correctly in the presence of mouse total RNA, our hybridization-based circuits might be adapted for in situ detection of complex expression patterns or even in vivo logic processing.

Acknowledgments

We thank Nadine Dabby for help with extensive revisions. Bernard Yurke built the custom fluorometer used for these experiments, and we are further indebted to him for inspiration and advice. GS was supported by the Swiss National Science Foundation and the Center for Biological Circuit Design at Caltech. EW acknowledges NSF awards #0093846 and #0506468, and HFSP award #RGY0074/2006-C. DS and DYZ were partially supported by an NIMH Training Grant to the Computation and Neural Systems option at Caltech. DYZ was partially supported by a Caltech Grubstake award.

Bibliography

- [1] T.S. Bayer and C.D. Smolke. Programmable ligand-controlled riboregulators of eukaryotic gene expression. *Nature Biotechnology*, 23(3):337–343, 2005.
- [2] Y. Benenson, B. Gil, U. Ben-Dor, and R. Adar. An autonomous molecular computer for logical control of gene expression. *Nature*, 429:423–429, 2004.
- [3] Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, and E. Shapiro. Programmable and autonomous computing machine made of biomolecules. *Nature*, 414(6862):430–4, 2001.
- [4] R.R. Breaker. Engineered allosteric ribozymes as biosensor components. *Current Opinion in Biotechnology*, 13(1):31–39, 2002.
- [5] R.M. Dirks and N.A. Pierce. From the Cover Triggered Amplification by Hybridization Chain Reaction. *Proceedings of the National Academy of Sciences*, 101(43):15275–15278, 2004.
- [6] M. Hagiya, S. Yaegashi, and K. Takahashi. Computing with hairpins and secondary structures of DNA. In *Nanotechnology: science and computation, Natural computing series*, pages 293–308. Springer, 2005.

- [7] F.J. Isaacs, D.J. Dwyer, and J.J. Collins. RNA synthetic biology. *Nature Biotechnology*, 24:545–554, 2006.
- [8] F.J. Isaacs, D.J. Dwyer, C. Ding, D.D. Pervouchine, C.R. Cantor, and J.J. Collins. Engineered riboregulators enable post-transcriptional control of gene expression. *Nature Biotechnology*, 22(7):841–847, 2004.
- [9] C.S. Lee, R.W. Davis, and N. Davidson. A physical study by electron microscopy of the terminally repetitive, circularly permuted DNA from the coliphage particles of *Escherichia coli*. *Journal of Molecular Biology*, 48(1):1–22, 1970.
- [10] M. Levy and A.D. Ellington. Exponential growth by cross-catalytic cleavage of deoxyribozymogens. *Proceedings of the National Academy of Sciences of the United States of America*, 100(11):6416, 2003.
- [11] D.E. Muller. Asynchronous logics and application to information processing. *Symposium on the Application of Switching Theory to Space Technology*, pages 289–297, 1962.
- [12] R. Penchovsky and R.R. Breaker. Computational design and experimental validation of oligonucleotide-sensing allosteric ribozymes. *Nature Biotechnology*, 23:1424–1433, 2005.
- [13] R. Sarpeshkar. Analog Versus Digital: Extrapolating from Electronics to Neurobiology. *Neural Computation*, 10(7):1601–1638, 1998.
- [14] G. Seelig, B. Yurke, and E. Winfree. Catalyzed relaxation of a metastable DNA fuel. *Journal of the American Chemical Society*, 128(37):12211–12220, 2006.
- [15] D. Sprinzak and M.B. Elowitz. Reconstruction of genetic circuits. *Nature*, 438(7067):443–448, 2005.
- [16] M.N. Stojanovic, T.E. Mitchell, and D. Stefanovic. Deoxyribozyme-based logic gates. *Journal of American Chemical Society*, 124(14):3555–3561, 2002.
- [17] M.N. Stojanovic and D. Stefanovic. A deoxyribozyme-based molecular automaton. *Nature Biotechnology*, 21(9):1069–1074, 2003.
- [18] A.J. Turberfield, J.C. Mitchell, B. Yurke, A.P. Mills Jr, M.I. Blakey, and F.C. Simmel. DNA Fuel for Free-Running Nanomachines. *Physical Review Letters*, 90(11):118102, 2003.
- [19] J. vonNeumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies*, pages 43–98, 1956.
- [20] B. Yurke and A.P. Mills. Using DNA to Power Nanostructures. *Genetic Programming and Evolvable Machines*, 4(2):111–122, 2003.
- [21] B. Yurke, A.P. Mills, and S. Lai Cheng. DNA implementation of addition in which the input strands are separate from the operator strands. *BioSystems*, 52(1-3):165–174, 1999.
- [22] B. Yurke, A.J. Turberfield, A.P. Mills Jr, F.C. Simmel, and J.L. Neumann. A DNA-fuelled molecular machine made of DNA. *Nature*, 406(6796):605–8, 2000.
- [23] Materials and methods are available as supporting material on Science Online at: <http://www.sciencemag.org/cgi/content/full/sci;314/5805/1585/DC1>.

Chapter 9

DNA as a Universal Substrate for Chemical Kinetics

Collaborators: Georg Seelig and Erik Winfree. **My contribution:** I developed the formulation of the goal, and came up with the construction. The text was written by me and GS.

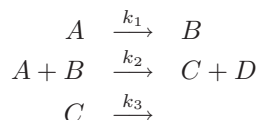
9.1 Abstract

We show that a DNA-based chemical system can be constructed such that it closely approximates the dynamic behavior of an arbitrary system of coupled chemical reactions. Using strand displacement reactions as a primitive we explicitly construct reaction cascades with effectively unimolecular and bimolecular kinetics. Our construction allows for individual reactions to be coupled in arbitrary ways such that reactants can participate in multiple reactions simultaneously, correctly reproducing the desired dynamical properties. Thus arbitrary systems of chemical equations can be compiled into chemistry. We illustrate our method on a chaotic Rössler attractor; simulations of the attractor and of our proposed DNA-based implementation show good agreement.

9.2 Introduction

Chemical reaction equations and mass action kinetics provide a powerful mathematical language for describing and analyzing chemical systems. For well over a century, mass action kinetics has been used to model chemical experiments, in order to predict and explain the evolution of the various species over time, and to elucidate the dynamical properties of the system under investigation. Chemistry exhibits complex behavior like oscillations, limit cycles, chaos or pattern formation, all of which can be explained by the corresponding systems of coupled chemical reactions [4, 7, 2]. While the use of mass action kinetics to describe existing chemical systems is well established, the inverse problem of experimentally implementing a given set of chemical reactions is typically much harder, and has not been solved in general. Many systems of coupled chemical equations appear to not have realizations in known chemistry.

Here we propose a method for implementing an arbitrary system of coupled chemical reactions using nucleic acids. We develop an explicit implementation of unimolecular and bimolecular reactions which can be combined into arbitrarily coupled reaction networks. In a formal system of chemical reactions such as



a species may need to participate in multiple reactions, both as a reactant and/or as a product (species A , B or C) and these reactions need to progress at rates determined by the rate constants (k_1 , k_2 and k_3). This

imposes onerous constraints on the chemical properties of the species participating in these reactions. For example, it is likely hard to find a physical implementation of the chemical reaction equations using small molecules, since small molecules have a limited set of reactivities. Information-bearing biopolymers such as proteins or nucleic acids provide a more promising physical substrate for implementing arbitrary chemical reactions. Nucleic acids have the unique advantage that interactions between different single-stranded species can be programmed since sequence determines reactivity through Watson-Crick base pairing.

In our DNA implementation, we assign each formal species (e.g., A, B, C, D) to a set of DNA molecules. In some instances it may be possible to map a formal species to a single oligonucleotide but more generally a single formal species will correspond to several DNA species in order to reproduce the correct kinetics. Effective interactions between the species are mediated by an additional set of DNA complexes. Since the underlying chemistry involves aqueous-phase nucleic acid hybridization and strand exchange reactions, arbitrarily large rate constants and concentrations cannot be attained. However, any system of coupled chemical reactions can be scaled to use smaller rate constants and concentrations without affecting the kinetics except by a scaling factor (see Section 9.7). While our constructions are purely theoretical at this point, they are based on realistic assumptions and provide a roadmap for future experiments.

In the next section we describe strand displacement reactions which will serve as the basic building block for our construction. In the following section we show how to implement arbitrary unimolecular reactions, and then extend our construction to cover bimolecular reactions. In the final section we give a demonstration of our approach on a system due to Willamowski and Rössler [9] with 3 species and 7 reactions exhibiting chaotic concentration fluctuations. Numerical simulations of the original formal system and our DNA-based chemical reactions using realistic rate constants and concentrations are in good agreement.

9.3 Cascades of Strand Displacement Reactions

Single-stranded nucleic acids with complementary sequences hybridize to form an inert double-helical molecule. Although hybridization reactions involve multiple elementary steps, for short oligonucleotides the kinetics is approximately a second-order process [1, 6]. However, hybridization between two complementary strands is insufficient to implement systems of coupled bimolecular reactions. For instance, the double-stranded product is inert, and thus incapable of acting as a reactant in another reaction.

Strand displacement reactions provide for a more promising primitive. The basic principle is illustrated in Fig. 9.1(b). Although a strand displacement reaction involves multiple elementary steps, likely including a random walk process, it is well modeled as a second-order process for a wide range of reaction conditions [5, 10]. The effective rate constant of the second-order process is governed by the degree of sequence complementarity between the toeholds on the single-stranded species and on the partially double-stranded species.

We have recently used strand displacement cascades to construct DNA-based logic circuits (Chapter 8). Here we use some of the nomenclature and ideas from that work. Fig. 9.2 shows a two-stage strand displacement cascade where an input single-stranded nucleic acid species (strand) initiates a strand displacement cascade between two complexes (gates) leading to the release of an output strand. In strand displacement cascades, a strand is functionally inactive before its release from a gate and becomes active upon becoming completely single-stranded. For example, intermediate strand o cannot react with translator gate t before it is released from gate g because its toehold domain 3, which is required for initiating the reaction with t , is double-stranded. Similarly, output Bs cannot initiate a downstream strand displacement cascade until it is released from translator gate t because its toehold domain 4 is double-stranded. However, upon the addition of free As , intermediate strand o is released through strand displacement, which then causes the release of output Bs .^{*} The release of strand Bs makes it capable of initiating other strand displacement cascades in turn. Multiple outputs can be produced by attaching two outputs to translator gate t and extending the intermediate strand o (as is shown in Fig. 9.3).

An input or output strand has two regions: a recognition region which can participate in a strand displacement reaction, and a history region which cannot. The sequence of the history region (e.g., domain 7 on strand Bs) is determined by the translator gate from which the strand was released. All strands with the same

^{*}The binding of a toehold domain to its complement is transient unless a strand displacement reaction can be initiated (in practice toehold domains are 2-10 nt long). Thus the 3 domain of input As does not block the 3* domain of translator gate t .

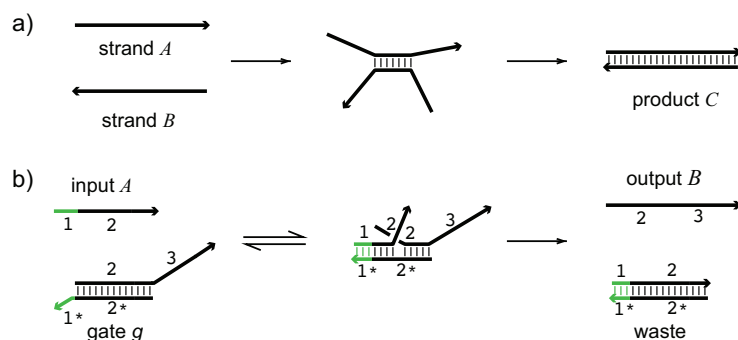


Figure 9.1: Hybridization and strand displacement reactions. **a)** Hybridization reaction. Two complementary strands A and B react with each other to form a double helix C . The hybridization reaction proceeds through a set of partially hybridized intermediates. Nevertheless, the overall reaction kinetics is well approximated as a bimolecular reaction $A+B \xrightarrow{k} C$. The 3' end of each strand is indicated by an arrow. **b)** Strand displacement. Functional sub-domains are numbered and the star indicates complementarity. The reaction between input strand A and gate g is initiated at the toe-hold (green, sub-domain 1*). The reaction then proceeds through multiple short-lived intermediates and leads to the release of an output strand B and the formation of a chemically inert double-stranded waste product. Kinetically, the overall reaction is well approximated as being bimolecular, i.e., $A + g \xrightarrow{k} B$, where we omit the inert waste product. The value of the rate constant k depends on reaction conditions (salt, temperature), length and sequence composition of the toe-hold as well as the degree of complementarity between the toe-holds on the strand and gate.

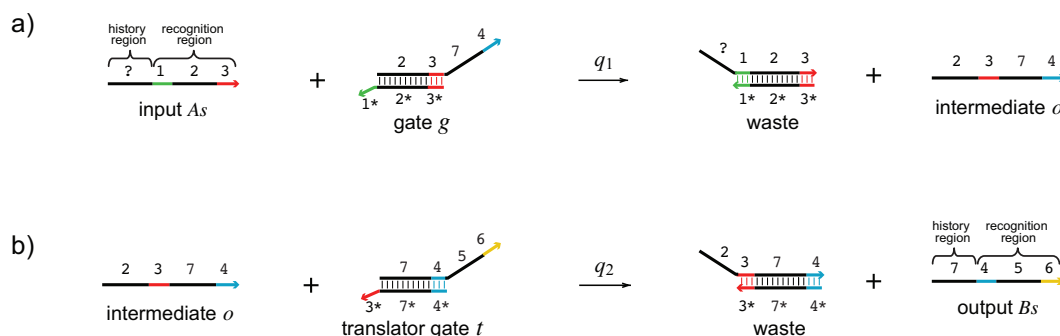


Figure 9.2: Two-stage strand displacement cascade. Functional domains are numbered and all toehold domains are indicated in color. Input or output strands with identical recognition regions react equivalently and are therefore grouped into the same species. For example, A_s is any strand with recognition domain 1-2-3, and B_s is any strand with recognition domain 4-5-6, irrespective of their history domains. The two-stage cascade shown produces B_s with history domain 7. Note that the sequences of the recognition regions of input and output strands A_s and B_s (domains 1-2-3 and 4-5-6) are completely unrelated to one another and therefore such a two-stage strand displacement cascade can link any input with any output species. **a)** Input strand A_s binds to gate g and by a strand displacement reaction releases the intermediate strand o . **b)** The intermediate strand o binds translator gate t and by a strand displacement reaction releases the output B_s .

recognition region react equivalently and we do not distinguish between them. For example, any strand with recognition domain 1-2-3 is called A_s and any strand with recognition domain 4-5-6 is called B_s . Since there are no sequence constraints (i.e., complementarity or equality) between the recognition region of the input strand A_s and the output strand B_s (similarly for multiple outputs as in Fig. 9.3), arbitrary chains of such two-step cascades can be linked together. This is possible for two-step cascades as shown; however, a one-step cascade would force a part of the recognition region of the output strand to have sequence equality with the input strand (see “full translator” in Chapter 8). We call the second gate a translator gate to emphasize its

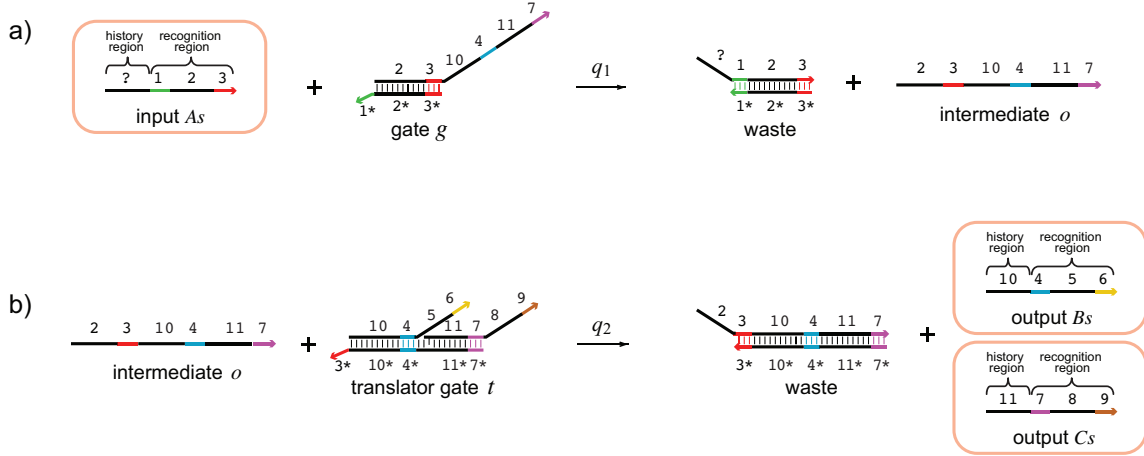


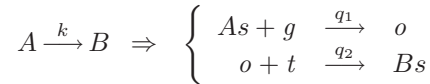
Figure 9.3: Molecular implementation of the unimolecular reaction $A \rightarrow B + C$. Orange boxes highlight the DNA species As , Bs , and Cs that correspond to the formal species A , B , and C . The sequences of the recognition regions of input and output strands As , Bs , and Cs (domains 1-2-3, 4-5-6, and 7-8-9, respectively) are completely unrelated to one another. The regime for desired unimolecular kinetics (concentrations of g , t and rate constants q_1 , q_2) is described in the text. **a)** Input strand As binds to gate g and by a strand displacement reaction releases the intermediate strand o . **b)** The intermediate o binds translator gate t and by a strand displacement reaction releases the outputs Bs and Cs .

role in translating the input to the appropriate output.

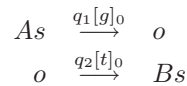
In the design of systems of coupled two-step cascades, nucleic acid sequences need to be constructed to avoid unintended interactions. For instance, we can first design all recognition regions to have maximally independent sequences, and then for every translator gate, design maximally independent history regions of its output strands.* Then a gate can react with only one recognition domain (g -type gates) or intermediate strand (translator gates), ensuring the specificity of interactions.

9.4 Arbitrary Unimolecular Reactions

As a first step we will implement the basic monomolecular reaction $A \xrightarrow{k} B$, such that A and B are single-stranded nucleic acid species with completely independent recognition regions. As we will show, the appropriate monomolecular kinetics can be obtained as a limiting case of the reaction kinetics for a two-step strand displacement cascade:



We use the notation As and Bs to mean the implementation of formal species A and B by DNA strands with recognition regions unique for A and B , respectively. We now discuss the conditions required to make the implementation valid. First, we will work in a regime where the concentrations $[g]$ and $[t]$ are in large excess of $[As]$ and $[o]$ so that they remain effectively constant at initial values $[g]_0$ and $[t]_0$ respectively. Then the two-step strand displacement cascade becomes equivalent to a pair of monomolecular reactions:



By varying the toehold strength of gate g which determines rate constant q_1 , or the initial concentration $[g]_0$, we set $q_1[g]_0$ equal to the formal rate constant k and attain $d[As]/dt = -k[As]$ as desired. To also

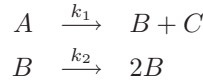
*In addition, all sequences must have minimal secondary structure, such as hairpin loops, because it can inhibit desired interactions.

ensure that $d[Bs]/dt = k[As]$, we make $q_2[t]_0$ large enough that intermediate strand $[o]$ settles to its quasi-steady-state value $q_1[g]_0[As]/(q_2[t]_0)$ on a much faster time scale than that on which $[As]$ changes. Then $d[Bs]/dt = q_2[t]_0[o] \approx q_1[g]_0[As] = k[As]$ as desired. To make the quasi-steady-state approximation hold in this example, we can increase the relative toehold strength of gate t compared to gate g , or use a much larger initial concentration $[t]_0$ than $[g]_0$.

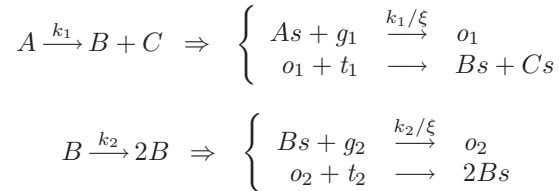
While experimentally, it may be useful to vary the degree of toehold complementarity affecting q_1 or concentration of gates $[g]_0$ to tune the effective rate constant, for simplicity throughout this paper we control reaction kinetics by tuning toehold strengths, while treating all gates as being present at the same high concentration ξ . Thus we set q_1 equal to k/ξ .

The same scheme can be extended to more complex unimolecular reactions. Reactions with more than one product species (e.g., $A \rightarrow B + C$ or $A \rightarrow 2B$) including catalytic (e.g., $A \rightarrow A + B$) and autocatalytic reactions (e.g., $A \rightarrow 2A$) can be constructed using a translator gate t that releases multiple strands as in Fig. 9.3. Removing the translator gate altogether allows for unimolecular decay reactions (e.g., $A \rightarrow$). Fractional product stoichiometry (e.g., $A \rightarrow (1/3)B + C$) can be realized using a mixture of translator gates with some fraction having incomplete output strands. For example, reaction $A \rightarrow (1/3)B + C$ can be implemented if 2/3 of translator gates t in Fig. 9.3 are missing the 7-8 domains.*

Arbitrary sets of unimolecular reactions can be coupled together by reusing the same recognition region in multiple reactions. Each reaction corresponds to a distinct two-step strand displacement cascade. For example, the system



can be implemented with gate-mediated reactions



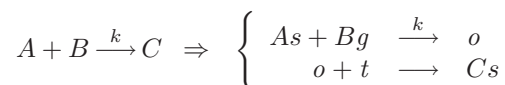
where unlabeled rate constants are much larger than k_1/ξ and k_2/ξ and initial concentrations $[t_i]_0, [g_i]_0 = \xi$ are high enough to remain effectively constant. The expressions for the DNA gate-mediated reactions in terms of formal rate constants are obtained from the above analysis. As described in the previous section, the different two-step strand displacement cascades do not have significant undesired interactions. Thus each reaction proceeds without interference from the others except through the desired coupling of input and output strands.

9.5 Arbitrary Bimolecular Reactions

Consider the basic bimolecular reaction $A + B \xrightarrow{k} C$. Since a reaction between an input strand and a gate can be viewed as being bimolecular, it provides a possible implementation of this reaction. As before, A is mapped to strand As , but now B would have to be mapped to a gate. To emphasize that a gate is mapped to a formal species B we call the gate Bg . As in the case of unimolecular reactions, we can use the translator gate t to ensure sequence independence between recognition regions of As and Cs . The corresponding

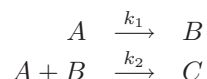
*Fractional product stoichiometries are equivalent to multiple reactions in which the same reactants produce different products, where the products are in integer stoichiometries. E.g. the two reactions $A \xrightarrow{2k/3} C$ and $A \xrightarrow{k/3} B + C$ are kinetically equivalent to a single reaction $A \xrightarrow{k} (1/3)B + C$. Conversely, all reactions with the same reactants but different products can always be combined into one reaction with possibly fractional product stoichiometries.

gate-mediated reactions therefore are:

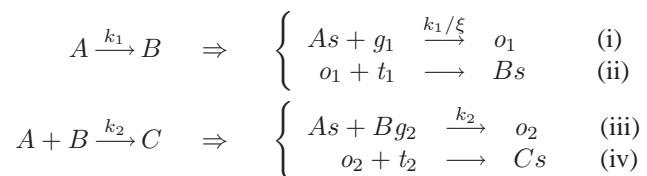


We set the unlabeled rate constant to be very large and the initial concentration of the translator gate $[t]_0 = \xi$ to be big enough to remain effectively constant. Then using the quasi-steady-state approximation for the intermediate strand o as in Section 9.4 we obtain the desired effective bimolecular reaction rate $k[As][Bg]$.

Having said that, this naive implementation has severe shortcomings. Since strand As must directly bind gate Bg , their sequences are not independent. Gate Bg can react only with input As and cannot participate in reactions with other strand species. Further, it is not always possible to uniquely assign reactants to a gate or a strand. One such example is the following system:

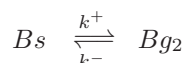


If we combine the implementation of monomolecular reactions developed in the previous section with the proposed bimolecular scheme, in the resulting system species B is mapped to two different forms, a strand Bs and a gate Bg_2 :



The concentrations of strand form Bs and gate form Bg_2 are entirely independent, and therefore the DNA reactions do not implement the desired formal chemical system.

However, if the two forms of B could be interchanged into one another on a time scale that is fast compared to the other reactions in the system, the correct behavior can be restored. We can link the two species Bs and Bg_2 through a fast reversible reaction



such that the two species achieve pseudoequilibrium. Then the formal species B exists in two different forms: $B = \{Bs, Bg_2\}$ and the total concentration of B is $[B] = [Bs] + [Bg_2]$. Let $f(Bg_2) = [Bg_2]/[B]$ be the fraction of B in gate form Bg_2 . Under the pseudoequilibrium assumption, $f(Bg_2) = (k^+ + k^-)/k^+$ is a constant. Since the second formal reaction can only use the gate form Bg_2 as a reactant, and not all of B , we scale the rate constant of reaction (iii) by $1/f(Bg_2)$ so that the new rate constant is $k_2/f(Bg_2)$. Then the effective rate of the implementation of $A + B \xrightarrow{k_2} C$ is $(k_2/f(Bg_2))[As][Bg_2] = k_2[A][B]$ as desired. We can easily extend this idea to create a pseudoequilibrium between strand Bs and gates Bg_i for multiple reactions i .

We realize the above reaction establishing pseudoequilibrium between Bs and Bg_2 via a linker gate shown in Fig. 9.4 (top). The mechanism corresponds to the following DNA reactions:



For the correct first-order kinetics $Bs \xrightleftharpoons[k^-]{k^+} Bg$, the linker gate l and the buffer strand b must be in excess, such that their concentrations remain effectively constant. Then $k^+ = q^+[b]_0$ and $k^- = q^-[l]_0$ where $[b]_0$ and $[l]_0$ are the initial concentrations of the buffer and linker strands respectively. For simplicity we will use $[b_0] = [l]_0 = \xi$ and $q^+ = q^-$.

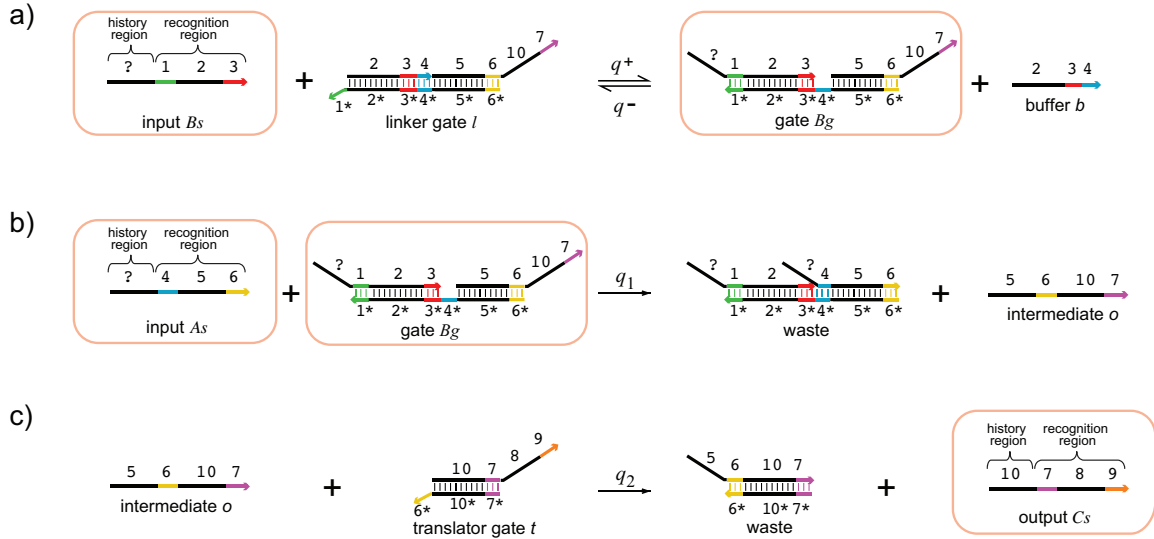


Figure 9.4: Molecular implementation of the bimolecular reaction $A + B \rightarrow C$. Orange boxes highlight the DNA species A_s , B_s , and C_s that correspond to the formal species A , B , and C . The sequences of the recognition regions of input and output strands A_s , B_s , and C_s (domains 1-2-3, 4-5-6, and 7-8-9, respectively) are completely unrelated to one another. The regime for desired bimolecular kinetics (concentrations of l , b , t and rate constants q^+ , q^- , q_1 , q_2) is described in the text. **a)** Input strand B_s reversibly binds to the linker gate l forming the activated gate B_g , i.e., $B + l \rightleftharpoons B_g + b$. **b)** Input strand A_s binds to the activated gate complex B_g and irreversibly releases intermediate strand o through strand displacement. **c)** The intermediate strand o binds translator gate t and by a strand displacement reaction releases the output C_s .

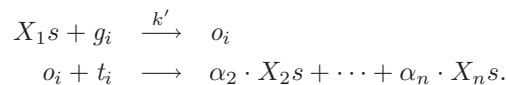
Lastly, we need to confirm the absence of unintended cross-reactions when implementing multiple coupled bimolecular reactions. As in the simple strand displacement cascades described in Section 9.3, gates can only react with specific recognition domains or intermediate strands. The exception to this rule is the reaction of gate B_g with the buffer strand b . Gate form B_g can react with any strand with accessible domains \dots 3-4. Because without loss of generality we can assume that there is only one formal reaction $A + B \rightarrow$ (see footnote on page 120), and domains 3 and 4 are unique to B_s and A_s respectively, nothing other than the correct buffer strand can react here.

9.6 Systematic Construction

In this section we take the ideas developed above and describe a systematic algorithm for compiling arbitrary unimolecular and bimolecular reactions into DNA gate-mediated chemistry. This algorithm is used in the next section to implement a Rössler attractor chaotic chemical system.

Without loss of generality we assume that every reaction has a unique combination of reactants (see footnote on page 120). Let i index reactions and $X_j \in \{A, B, C, \dots\}$ index species. Let $f(X_j s)$ be the fraction of X_j in strand form $X_j s$. Similarly let $f(X_j g_i)$ be the fraction of X_j in gate form $X_j g_i$.

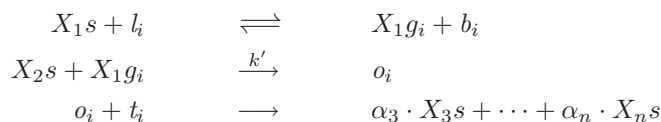
Consider any unimolecular formal reaction i . Write the reaction as $X_1 \xrightarrow{k} \alpha_2 \cdot X_2 + \dots + \alpha_n \cdot X_n$, where $\alpha \leq 1$. We implement this reaction by a two-step strand displacement cascade (Fig. 9.3), modeled by the DNA gate reactions below (where we omit inert waste products, and combine all strands with the same recognition domains into a single species).



Product fractions α_j are set by preparing translator gate t_i with α_j fraction of complete and $1 - \alpha_j$ incomplete

output strands for $X_j s$ as discussed in Section 9.4. Unlabeled rate constants as well as the initial concentrations $[g_i]_0 = [t_i]_0 = \xi$ are as high as possible. Rate constant k' is set to $\frac{k}{\xi f(X_1 s)}$ by varying the degree of complementarity of the toehold on gate g_i with the toehold on strand $X_1 s$. Note that by following the argument of Section 9.4, and using the fact that $[X_1] = [X_1 s]/f(X_1 s)$, the effective rate of this reaction is $k'[X_1 s]\xi = k[X_1]$ as desired.

Consider any bimolecular formal reaction i . Write the reaction as $X_1 + X_2 \xrightarrow{k} \alpha_3 \cdot X_3 + \dots + \alpha_n \cdot X_n$, where $\alpha \leq 1$. We implement this reaction by a linker gate mechanism combined with the two-step strand displacement cascade (Fig. 9.4) and is modeled by the DNA gate reactions below (where we again omit inert waste products, and combine all strands with the same recognition domains into a single species).



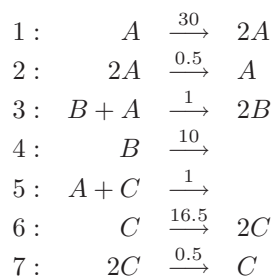
Product fractions α_j are set by preparing translator gate t_i with α_j fraction of complete and $1 - \alpha_j$ incomplete output strands for $X_j s$ as discussed in Section 9.4. Unlabeled rate constants are as high as possible, with the forward and reverse rates of the first reaction being equal. Rate constant k' is set to $\frac{k}{f(X_2 s)f(X_1 g_i)}$ by varying the degree of complementarity of the toehold on $X_1 g_i$ with the toehold on strand $X_2 s$. The initial concentrations $[l_i]_0 = [b_i]_0 = [t_i]_0 = \xi$ are as high as possible. Note that by following the argument of Section 9.5, and using the facts that $[X_2] = [X_2 s]/f(X_2 s)$ and $[X_1] = [X_1 g_i]/f(X_1 g_i)$ the effective rate of this reaction is $k'[X_2 s][X_1 g_i] = k[X_1][X_2]$ as desired.

With the above construction, determining $f(X_j s)$ and $f(X_j g_i)$ is easy: for every i, j , $f(X_j s) = f(X_j g_i) = 1/(m + 1)$ where m is the number of bimolecular reactions in which X_j appears as the first reactant.

The sequences of the DNA components can be designed as follows. First, for every species X_j , design an independent recognition region. Then, for each formal reaction, design independent history regions for every output of that reaction. At this point all auxiliary DNA species are fully specified. Significant unintended interactions between auxiliary species participating in different formal reactions cannot occur by the arguments in Sections 9.3 and 9.5. The system is initiated by adding appropriate starting amounts of the formal species in single-stranded form with arbitrary history domains.

9.7 Example

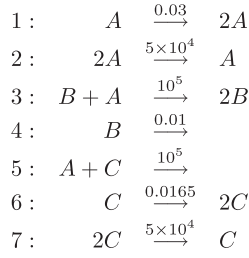
We illustrate our method of using DNA-based chemistry to implement arbitrary formal systems of coupled chemical equations on the chaotic system due to Willamowsky and Rössler [9]. We start with the following formal reactions, where the rate constants are from Ref. [3]:



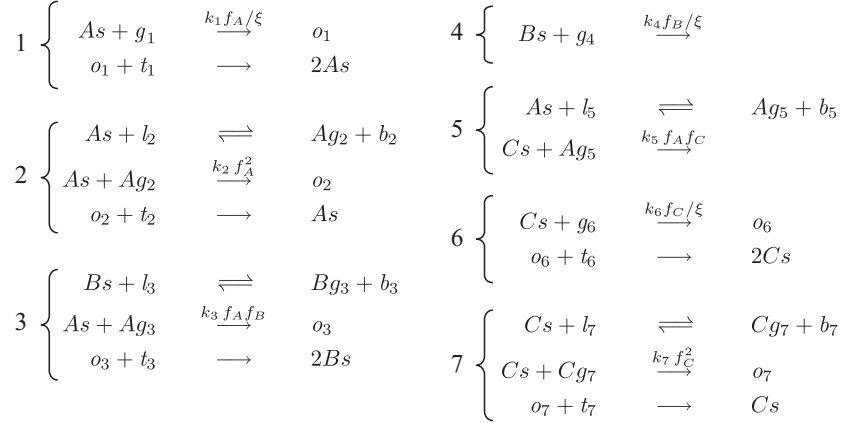
The strange attractor for the concentrations of A , B , and C is in the range of about 0–40.

First we scale this system into a regime realistic for DNA-based chemistry which constrains reaction rates and concentrations. Second order rate constants for strand displacement reactions can be approximately in the range $0-10^6/M/s$, with their value determined by the degree of toehold complementarity [10]. Typical experimental concentrations are on the order of $0-10^{-3}M$. Similar to experimental implementations of other dynamical chemical systems, a flow reactor may be used to replenish the stock of unreacted gates and remove waste to maintain the appropriate reaction conditions [2]. This may make it possible to use lower

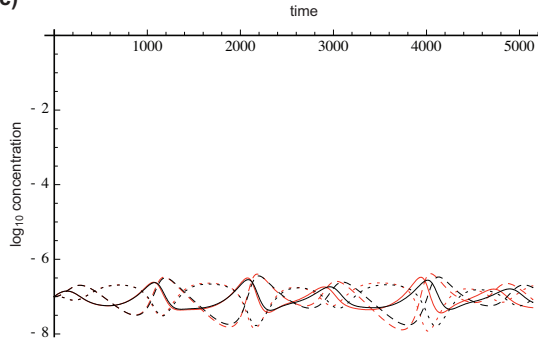
a) Original system



b) Reactions for DNA implementation



c)



d)

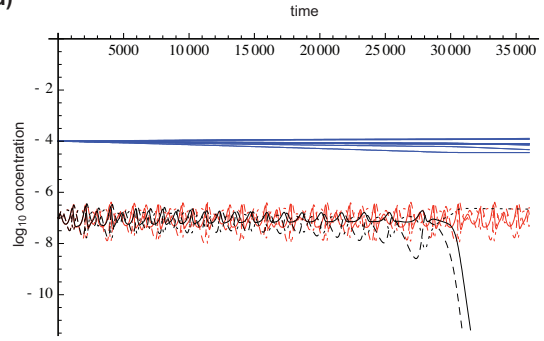


Figure 9.5: Rössler attractor example. **(a)** The formal chemical reaction system to be implemented. **(b)** Reactions modeling our DNA implementation. Each bracket implements the formal reaction with the number indicated. Here k_1 through k_7 are the original rate constants for reactions 1 through 7 as in (a). Multiplicative factors $f_A = 1/f(As) = 1/f(Ag_2) = 1/f(Ag_5) = 3$, $f_B = 1/f(Bs) = 1/f(Bg_3) = 2$, $f_C = 1/f(Cs) = 1/f(Cg_7) = 2$. We use initial concentration of the gates and buffer strands $\xi = 10^{-4}$. Unlabeled rate constants are 10^5 . **(c)** Plot of the log-concentrations of A (solid), B (dashed), C (dotted) for the original system (red), as well as their modeled concentrations (black). **(d)** Longer time plot showing also the log-concentrations of g_i (blue, decreasing) and b_i (blue, increasing). **(e,f)** Trajectories of the original system and DNA implementation in the 3-dimensional phase-space (first 5 hours).

gate concentrations.

Clearly, by scaling all rate constants by the same factor we simply speed up or slow down the system without affecting the dynamical behavior. We can scale the concentrations at which the chaotic system operates by scaling the bimolecular rate constants differently from the unimolecular ones. In general if $[X_j](t)$ are solutions to differential equations arising from a set of unimolecular and bimolecular reactions, then $\alpha[X_j](t)$ are solutions to the differential equations arising from the same set of reactions but in which we divide all bimolecular rate constants by α . We first slow down the system by multiplying all rate constants by 10^{-3} , and then use concentration scaling factor $\alpha = 10^{-8}$, obtaining the rate constants in Fig. 9.5(a).

Applying our construction yields a DNA implementation governed by the equations in Fig. 9.5(b). Simulations confirm (Fig. 9.5(c, d)) that the DNA implementation behaves very close to the formal system (a) until the depletion of linker gates l_i and the buildup of buffer strands b_i sufficiently alters the effective rate constants to destroy the chaotic behavior at around 8 hours (see (d)).

9.8 Conclusion

We have proposed a method for approximating an arbitrary system of coupled unimolecular and bimolecular chemical reactions using DNA-based chemistry. Our construction takes advantage of cascades of strand displacement reactions (Chapter 8), and elementary techniques of approximation in chemical kinetics. Each formal species occurring in the system of chemical reactions is represented as a set of strands and gates. The multiform representation is necessary because it is not always possible to find a single DNA species that is capable of participating in all reactions involving a given formal species. However, the different forms are constructed to be in equilibrium with each other and thus participate in kinetics as if they were a single species, up to a scaling of rate constants.

While we have taken care to provide a systematic algorithm for compiling a set of chemical reactions into DNA, in practice it may often be possible and preferable to reduce the complexity by optimizing the construction for the particular system of interest. For example, in many cases complete sequence independence between strands may not be necessary, possibly allowing one to eliminate some translator gates. Similarly, pseudoequilibrium linkage is unnecessary if mapping a species directly to a strand or gate does not cause problems.

Acknowledgments

This work was supported by NSF Grant 0728703. We thank D. Zhang, J. Schaeffer, and M. Magnasco for useful discussions.

Bibliography

- [1] C. R. Cantor and P. R. Schimmel. *Biophysical Chemistry, Part III: The behavior of biological macromolecules*. W. H. Freeman and Company, 1998.
- [2] I. R. Epstein and J. A. Pojman. *An Introduction to Nonlinear Chemical Dynamics: Oscillations, Waves, Patterns, and Chaos*. Oxford University Press, 1998.
- [3] P. Gaspard. *Encyclopedia of Nonlinear Science*, chapter “Rössler Systems”, pages 808–811. Routledge, 2005.
- [4] G. R. Gavalas. *Nonlinear Differential Equations of Chemically Reacting Systems*. Springer-Verlag, 1968.
- [5] C. Green and C. Tibbetts. Reassociation rate limited displacement of dna strands by branch migration. *Nucleic Acids Research*, 9:1905–1918, 1981.
- [6] L.E. Morrison and L. Stols. Sensitive fluorescence-based thermodynamic and kinetic measurement of dna hybridization in solution. *Biochemistry*, 32:3095–3104, 1993.

- [7] S. K. Scott. *Chemical Chaos*. Oxford University Press, 1991.
- [8] G. Seelig, D. Soloveichik, D. Y. Zhang, and E. Winfree. Enzyme-Free Nucleic Acid Logic Circuits. *Science*, 314(5805):1585–1588, 2006.
- [9] K. D. Willamowski and O. E. Rössler. Irregular oscillations in a realistic abstract quadratic mass action system. *Zeitschrift für Naturforschung A*, 35:317–318, 1980.
- [10] B. Yurke and A. P. Mills. Using DNA to Power Nanostructures. *Genetic Programming and Evolvable Machines*, 4(2):111–122, 2003.