THEORY AND EXPERIMENTS IN ALGORITHMIC SELF-ASSEMBLY

by

Paul Wilhelm Karl Rothemund

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

December 2001

Acknowledgments

An advisor, I am fond of telling those who are not familiar with graduate school and express interest in the phenomenon, can be likened to some complicated combination of a parent and a significant other. You love them, you hate them, you wish to please them, you rebel against them, you want to grow up just like them, you hope to hell you never treat your children the same way, you miss them dearly and you just need more space. In the end, as with your parents, you end up owing them a debt that can never be repaid except by raising intellectual children of your own and passing the debt on to them. Len Adleman, number one, has been an excellent advisor.

Gilgamesh had Enkidu, and so I have Erik Winfree, except the relationship is perhaps vice versa because Erik is more of an epic hero, and I more a forest wild man. For better or for worse, we jumped into the whirlpool of DNA computing together — and more than once Erik has pulled my head above water. Erik is a source of inspiration and a mirror into which I look, upon occasion, to learn something about myself.

As for the rest, within the scientific community and without, who helped directly with my thesis, provided moral support, or helped maintain balance in my life — I like to consider them an extended family.

Every family has elders. For a graduate student they are his committee members: Ashish Goel, Aristides Requicha, Ming-Deh Huang, and Myron Goodman. Myron, in particular, has offered much practical advice about graduate school and off-color humor.

A graduate student's nuclear family is his group. I have great affection for the group Len has assembled: Nickolas Chelyapov, Ravi Braich, Areio and Ali Soltani, Darryl Hwang, Clifford Johnson. I have learned much about experimental technique, as well as fine art, from Nickolas.

For perspective I thank a pair of favorite uncles, both chemists: Nadrian Seeman, terror of poultry everywhere, and Randy Lee, prodigal son of Dangerfield, Texas.

Then there are numerous other cousins, aunts, and uncles to thank, some of them black sheep, all of them characters: Dori Levanoni, Karen Casciotti, Michael Su, Rebekah Mills, Deborah Fygenson, Anthony Leonardo, Roian Egnor, Hideo Mabuchi, Matt Cook, Andrea Foegler, Joe Demers, Mike Hartenstine, Vicki Brown, Brigette Tippin, Karen Nordell, Sharon Laubauch, Penelope Sherman, Christine Frank, Sanjoy Mahajan, Sam Roweis, Andreas Guazelli, John Petruska, Joe Kirschvink, Robert Creswell, Preston Pfarner, Lisa O'Rourke, Bill Craven, Ned Bowden, George Whitesides, Lior Pachter, Moeen Abedin, John Baldeschwieler, John Hopfield, Laura Rodriguez, Bernie Yurke, Khinh Bui, Qi Cheng, David Kempe, Rob Rossi, Nick Papadakis, Yessenia Rivera and many others.

A remarkable branch of the family is the DNA computing community: Grzegorz Rozenberg, Lila Kari, Harvey Rubin, David Wood, David Gifford, Richard Lipton, Anne Condon, Laura Landweber, John Reif, John McCaskill, Tom Knight. They provided me opportunities to travel across the country and to foreign lands to see the inside of Antiquities museums, North Sea beaches, and Boston from the back of a Duck.

A respectable branch of the family includes teachers: Kate Reardon, Aditi Dhagat, Barun Chandra, Amitabh Shah, J.L.A. van de Schnepscheut, Mani Chandy, Yaser Abu-Mostafa, Irene Wright, Bela Julesz, Greg Fu, Elizabeth Burns, Christoph von der Maalsburg, Michael Arbib, and Tom Apostol. I thank them for teaching me about math, computer science, psychophysics, chemistry, and brains.

A group of long lost relatives, rediscovered just a few years ago, is the circle of boomerang throwers: Southbay Johnny, Erik Fields, Gel Girvin, Kelly Andretti, Dan Neelands, Steve Conaway, Rainer Graebenteich and Jim Mayfield. They introduced me to Sprinklerhead, Radness, GLORP, Long Distance Contests, Kiel, and Yanakis.

Similarly spirited relations are the sailplane pilots: Nowell Siegel, Joe Wurts, Dave Sanders, John Roe, Charlie Neuman, Larry Jolly, Steve Fujikawa, Jerry Teisan, and Rudy. They provided me with lessons from World Champions, combat contests, midnight flying, Zagis, midairs, vistas of the Wasserkuppe, and many opportunities to scale down cliffs.

For administrative assistance I thank Amy Yung, Julieta dela Paz, Bende Lagua, and Maria McElwee. Without them, no classes would have been registered for, and no money would have been spent.

For technical help I thank John Lyons and Lynn Bennett of Laser Connection, Flag's Photo, Solter Plastics, and Alicia Thompson.

For financial support I thank the following funding agencies: DARPA, the Sloan Foundation, the Office of Naval Research, the National Science Foundation, and NASA/JPL.

Finally I thank all my biological family, especially my sister Julie, and my parents Max and Judith. They nurtured my interest in science and the world from the very beginning — I thank them for indulging these interests in one funny little kid.

Contents

List Of Tables

List Of Figures

Abstract

Self-assembly plays a central role in the organization of matter; out of basic units ranging from atoms to stars, self-assembly creates complex patterns ranging from crystals to galaxies. Investigations of DNA computing have revealed a fundamental connection between self-assembly and computation: in principle, any computation can be performed by a suitable self-assembling system.

Here, we study this connection both theoretically and experimentally. We begin with a theoretical study of square tiles, known as Wang tiles, using the Tile Assembly Model (TAM). The TAM models temperature and other experimental conditions using an integer parameter $\mathcal{T}$. We study the TAM at $\mathcal{T} = 1$, under which tiles may bind to an aggregate via single bonds, and at $\mathcal{T} = 2$ under which tiles must bind by at least two bonds. For $\mathcal{T} = 2$ we find that an efficient $O(\log N)$ complexity can be achieved whereas for $\mathcal{T} = 1$ we prove that the complexity is exactly N^2. The exponential improvement for $\mathcal{T} = 2$ is achieved by simulating a binary counter; hence, while universal computation can be performed by self-assembly under both conditions, it appears that computation can be used to greater effect at $\mathcal{T} = 2$.

Real systems likely lie between $\mathcal{T} = 1$ and $\mathcal{T} = 2$; finding systems that approximate $\mathcal{T} = 2$ is an important goal — such systems would open the door for the use of computation in self-assembly. Toward this end we explore an experimental system in which plastic tiles

self-assemble via capillary forces. We test the system's ability to compute using a set of "XOR" tiles that would simulate a well-known cellular automaton if $\mathcal{T} = 2$ but would assemble with errors if $\mathcal{T} = 1$. Our experiments show that the XOR tiles assemble with few errors; thus they exhibit $\mathcal{T} = 2$-like behavior.

Finally, we begin to explore error rates and nucleation in a DNA-based system. We design DNA tiles to form 2D lattices bearing a pattern of stripes readily visible under the atomic force microscope. Error-free ($\mathcal{T} = 2$) growth would result in a pattern of continuous stripes; errors would result in broken stripes. Initial experiments show that errors can be visualized allowing future studies to optimize conditions for $\mathcal{T} = 2$ growth.

Chapter 1

Introduction

In theory there is no difference between theory and practice; in practice, there is.

—J.L.A. van de Schnepscheut

1.1 What is Self-Assembly?

Consider, for a moment, the manufacture of a car. This might seem a daunting task — a car is a pretty complex machine. It has thousands of bits of interlocking metal, plastic, and glass that must all be fit together in the proper manner.

Now imagine that rather than manually fitting all of the parts together, one applied glue to the surfaces of the parts, dumped them in a giant box, and shook the box. Peering inside, one could hardly expect to see a car; parts would, it seems, stick to each other willy-nilly resulting in a modern sculpture rather than a car. And yet ... what if one were clever about the shapes of the parts so that two colliding parts could be joined only if they fit each other perfectly? Or what if one had more than one type of glue, so that two colliding parts could be joined only if they shared the same type of glue? Then might

such an approach to manufacturing cars be possible? Could such an approach be used for manufacturing in general?

For cars it seems unlikely; glass windshields and heavy engine blocks are probably incompatible in a roiling soup of car parts. But amazingly, just this sort of spontaneous unsupervised process, known as *self-assembly* is how many structures in the natural world are built: Atoms stack in periodic arrays to form crystals. Complementary strands of DNA wind around each other to form a double helix. Phospholipids associate to form bilayers, micelles, vesicles and more complicated structures. Viral coat proteins fit together around a nucleic acid to form a viral particle.

These molecular examples are generally accepted as instances of self-assembly. But no precise definition of self-assembly seems to exist, and scientists differ on exactly which processes to classify as self-assembly. Some would use it only for molecular processes; some only if binding is "reversible"[1] or "weak" or "noncovalent" [WMS91]; some only if the movement of objects is dictated by thermal agitation. Others use self-assembly more liberally[2]: for micron-sized latex spheres that crystallize based on entropy [KRYP94, DYP95], for centimeter-sized plastic shapes that associate by capillary forces [BTCW97], or even for planet-sized and larger objects that affect each other through gravity.

[1] In this thesis we attempt to adhere to the following rules: Words being defined as part of a mathematical formalism are in **bold face**. Words being informally defined, words perhaps unfamiliar to the reader, and words being stressed are *italicized*. Real and manufactured quotes, metaphors, labels, multi-word concepts and words that are controversial or being slightly abused are "in quotes".

[2] George Whitesides, considered an expert in self-assembly for his work on self-assembled alkanethiol monolayers on gold, is in this camp. He defines self-assembly as processes for which the attractive force between objects is "mesoscale", that is, operating at length scales about the same size as the objects themselves. But this definition is a little unsatisfactory since mesoscale is a fuzzy term, itself subject to interpretation: Whitesides includes celestial mechanics under the umbrella of self-assembly but, clearly, heavenly bodies orbit each other at distances thousands of times their own diameter. Adding to this confusion, physicists and atmospheric scientists have their own very different definitions for mesoscale. Thus we choose not to use "mesoscale" in our description of self-assembly.

Given this confusion we make no formal definition for self-assembly in general. Informally, we find it useful[3] to think of self-assembly as the aggregation of immutable objects under a static (non-time varying) attractive potential which provides the only energy used. Also part of such a system is some kind of random motion, analogous to thermal motion, that helps the system relax toward equilibrium. The magnitude of this motion, analogous to temperature, dictates whether the attraction between objects can hold them in a bound state. In this thesis, we limit our study of self-assembly to systems that fit this informal notion, and further restrict it to a crystallization-like system whose basic units may be abstracted as rigid square tiles known as Wang tiles. So that we can prove statements about self-assembly we give, in Chapter 2, a formal model for what it means to self-assemble Wang tiles.

1.2 Algorithmic Self-Assembly

Self-assembly in natural systems has long fascinated physical scientists. They ask fundamental questions like "How should order be defined for self-assembled structures?" or, "By what mechanisms do complex structures self-assemble from simple units?" Further, as the succession of photolithographic technologies that have allowed us to structure matter at smaller and smaller length scales runs out, engineers are beginning to recognize self-assembly as a powerful method for creating patterns: Molecular self-assembly naturally

[3]Useful because it helps maintain self-assembly as a meaningful term, distinguished from self-organization, chemistry, or physics; it allows us to exclude some phenomena from self-assembly, and focus our study. For example, the copying of DNA by a polymerase we consider a more complicated chemical process, one driven forward by additional energy from hydrolysis of nucleotide triphosphates. Likewise, the development of a multicellular organism, in which cells divide, differentiate, and may actually migrate we consider an instance of *self-organization* rather than self-assembly.

yields atomically precise structures[4]. Engineers ask: "How can a self-assembling system be designed to yield a desired pattern?" Thus the development of a "theory of self-assembly" has become important both to an understanding of natural pattern formation and for realizing self-assembly's engineering promise. It has become clear, however, that current theories cannot fully address our questions about self-assembly. To see this consider just our first question, that of defining "order".

Over the last century the dominant and most successful theoretical framework for understanding order in chemical self-assembling systems has been the discipline of "mathematical crystallography" [Sen90]. Treating order as synonymous with "periodic order" and holding symmetry as its defining character, crystallographers used elementary group theory and geometry to enumerate all possible symmetry groups for periodic structures in 3-space. From their enumeration in 1890 until the 1980s these "space groups" essentially defined order: the term crystal was reserved for materials characterized by one of the 230 space groups; everything else was described as disordered, amorphous, or glassy.

Then, in 1984, quasicrystals were discovered [SBGC84]. Macroscopically faceted and microscopically regular enough to yield sharp spots under diffraction, these structures fit our intuitive notion of "ordered material". And yet they had five-fold symmetry — a symmetry that is "forbidden" to periodic structures — and hence could not be assigned to one of the space groups. Thus, quasicrystals broke the monopoly of the space groups but left a quandary still pondered today — what is order if not periodic? An answer that acknowledges the order inherent in quasicrystals is simply to redefine crystal as "a

[4]Even if other new methods for patterning atoms (*e.g.* scanning probe manipulation) can be mastered, self-assembly, because it is inherently parallel, is likely to offer a cheaper and faster alternative in cases where it works.

structure with an essentially discrete diffraction pattern" [Sen95]. But such a patch for the existing framework seems unsatisfactory; it leaves little room for still more exotic structures that may lurk undiscovered and excludes altogether the order we intuitively recognize in complex biological materials.

These concerns have led the crystallographer Alan Mackay to propose that a "generalized crystallography" might define order [Mac95] using computer programs and cellular automata. A natural proposal[5] is to define the order in self-assembled object in terms of the size of the smallest computer program that could print out a list of the identity and coordinates of every atom in the object — essentially the Kolmogorov complexity. The smaller the computer program (in relation to the size of an object) the more "ordered" the object. This definition of order does not divide materials into ordered and disordered as does a definition based on periodicity and nonperiodicity; instead it seems better, capturing gradations in order that we recognize in real materials. Truly periodic structures have very small programs — essentially describing just the smallest repetitive structure (unit cell) and size of the crystal — and hence have high order. Any nonperiodic structures that, like quasicrystals, may also be described by very small programs are, as we might naturally desire, also considered to be highly ordered. "Nearly" periodic structures bearing a few random defects need somewhat larger programs: their programs may describe the original periodic structure plus a list of the few places it has defects; hence they have less order. Random aggregates of atoms require large programs describing the explicit placement of every atom in the aggregate and are thus maximally disordered.

[5] One that in my experience occurs often to computer scientists.

Such computational descriptions for self-assembly are not limited to order — growth mechanisms, kinetics of growth, and complex mixtures of self-assembled molecules all have similar computational descriptions. Taken together, such descriptions form a computational framework for studying self-assembly, *algorithmic self-assembly*, that is attractive for two reasons[6]. First, assuming that physics is just a model of computation, it is a corollary of Church's thesis that computer programs will be able to capture all of the complex behavior of self-assembly — no more complicated theory will be required. Second, such a framework may allow principles of theoretical computer science to be translated into statements about the physical world. For example, the self-assembly of DNA structures may be mapped naturally onto the languages of the Chomsky Hierarchy [WYS98]: linear DNA self-assembly is equivalent to the regular languages, assembly of DNA dendrimers (trees) is equivalent to the context-free languages, and assembly of four-connected DNA lattices is equivalent to unrestricted grammars.

Algorithmic self-assembly, as a theory for self-assembly, will ultimately be judged by its successes — interesting predictive or explanatory results in natural self-assembly or constructive results for engineering. Attempts to make well-defined computational descriptions for physical processes are not new[7] and seem to form a promising basis for a physical theory. Yet computational measures for physical complexity claim no great successes and have not generated wide interest outside of computer science and a subset of the

[6]We note that the use of an algorithmic framework for the study of any type of self-organization or complex physical system is attractive for the same two reasons.

[7]Particularly interesting are some measures of complexity that take into account not only the size of a program required to create a string, but also the time. For such measures both simple periodic patterns and random ones have low complexity; only patterns that are truly difficult to create (because they require lots of time) are assigned high complexity. Chaitin, Levin, Adleman, and Bennett have all studied such complexities in recent years [Cha77, LV77, Adl79, Ben85, Lev84, Ben95].

physics community that studies cellular automata (for examples, see [Wol94].) Possible reasons for this include a lack of education on the part of physical scientists[8] and the general intractability of computational theories[9]. Less interesting (perhaps disturbing?) to the computer scientist is the possibility that aside from human brains and silicon computers, physical systems do not do much non-trivial computation that can be analyzed with their methods. The algorithmic point of view may be sufficient to describe every possible phenomena in self-assembly but in practice is it necessary? A less powerful and more tractable theory for describing self-assembly might be sufficient, and if possible, would be desirable.

Computer scientists can, for the moment, breathe easy — it appears that the full theory of computation is necessary: not only do there exist, in principle, self-assembling systems capable of universal computation but we have a candidate physical system that may allow us to implement computations practically. This first result in algorithmic self-assembly has its roots two disciplines: the mathematical theory of tiling and DNA computation. In 1963 Hao Wang showed that the simulation of Turing machines could be embedded in tilings[10] of the plane by squares [Wan63]. In 1995 Erik Winfree showed that, in general, Turing machine tilings[11] could be realized by the physical process of crystallization

[8]Biologists still explore *ad hoc* measures of complexity like "the number of cell types" [Bon88]. The recent availability of full genomic sequences may allow more rigorous complexity measures to be explored.

[9]Computational theories are in general much more complicated than theories with finite results like mathematical crystallography. For example, the "computational order" that we describe above is really just the Kolmogorov complexity of a string and it is, in general, uncomputable. Nevertheless Kolmogorov complexity is not a useless concept; computer scientists move happily along proving bounds on the complexity for many strings of practical interest.

[10]A *tiling* is a covering of the plane by geometric shapes without gaps or overlaps. A tiling is an abstract mathematical object, infinite and without defects — no physical mechanism for its generation is specified. When we say that a tiling may be grown by self-assembly, we mean that an arbitrarily large *partial tiling* consistent with the infinite mathematical tiling may be grown.

[11]Turing machine tilings may be grown by self-assembly using the methods of Winfree. But Winfree actually described the simulation of one-dimensional cellular automata by self-assembly rather than Turing

and proposed that, in particular, DNA self-assembly would be a good substrate for their implementation[12] [Win96].

On the surface Winfree's contribution is trivial — just a simple correspondence between Wang's tilings and two-dimensional self-assembly. Wang's construction, however, gives no mechanism for how the tilings are to be made; rather it speaks only to the existence of tile sets and tilings that simulate Turing machines. Winfree's work, in contrast, gives a detailed mechanism for the creation of tilings and addresses whether the mechanism is physically plausible. This difference is important since the question of whether it makes any sense to model self-assembly as an algorithmic process hinges on it: we are *forced* to deal with the complexities inherent in universal computation and *allowed* to reap its benefits *if and only if* self-assembling systems can perform universal computation *in practice*.

Further, while Winfree's proposal is promising, its reduction to practice is by no means a *fait accompli*. For several reasons, both theoretical and practical, we expect that the translation of Wang's theory into a physical implementation will not be straightforward and will necessitate the development of new theory and experimental technique:

---

machines. Such cellular automata are equivalent to Turing machines in terms of the functions they can compute but are, in general, much faster. Thus if the reader thinks cellular automata are a more interesting or more practical model of computation, she should feel free to add "or cellular automata" to every occurrence of "Turing machines" in this chapter. Whether a characterization of the full power of self-assembly will reveal it to be a significantly faster model of computation than cellular automata is unknown but, to me, seems unlikely.

[12] In fact computation by self-assembly predates this result; one step in Adleman's original algorithm for the Directed Hamiltonian Path Problem [Adl94] uses linear DNA self-assembly to create DNA molecules to represent all paths through a graph. Implicit in this step is the equivalence of linear self-assembly to the generation of regular languages. Even if universal computation by 2D self-assembly is never practical, this connection between computation and self-assembly is firm — linear self-assembly of DNA works fairly well in practice (as well can be expected of chemistry, anyway). Simple regular languages of DNA molecules are used wherever diverse populations of molecules are required — in many laboratory procedures for mutagenesis and *in vitro* evolution, as well as in our immune systems. Outside of schemes for DNA computation, however, uses for the most general regular languages of chemical structures are not yet known.

(i) The basic tiles used in Turing machine tilings are much more complicated than the basic units found in natural crystals. The tiles typically have no rotational or reflective symmetries; often all four bonds are of different types. The asymmetric synthesis required to generate such tiles as molecules is a difficult synthetic challenge. Further the number of distinct tiles needed (often half a dozen or more) adds greatly to the synthetic effort required. So far, the only type of molecule that has shown promise for implementing Turing machines is DNA; the design of the necessary DNA motifs, however, is still a new and developing science [See98, SWY+98].

(ii) The tilings themselves are, in general, more complicated than the periodic structures of natural crystals. Typically[13], for a set of tiles that simulates a Turing machine, the identity of a tile at a particular site depends on two or more of its neighbors. Simulations [Win98a] suggest that crystal growth can enforce these dependencies, albeit probabilistically, under certain physical conditions.[14] This introduces a concept of *error rate* not present in the theory of tilings. Also, tiles used to simulate Turing machines are typically consistent with tilings other than the desired simulation. Mathematically, we constrain tiles to form a desired structure merely by defining valid structures as those that contain certain *seed tiles* (See Chapter 2). The physical world cannot be so constrained; we must learn how to bias the *nucleation* (initial growth) of structures so that only the desired structures form, or form at a higher rate. More subtly, there is reason to believe that

[13]That is to say, the Turing machine (and cellular automata) constructions we consider most seriously for implementation in physical self-assembling systems have this property. Other sets of tiles for Turing machines that don't have this property have other undesirable properties; we are unsure what properties are strictly necessary for tile sets that compute. We discuss this more in the next section and Chapter 2 Section 2.6.5.

[14]In Chapter 3 we briefly meet even stranger tilings, Penrose tilings, for which the identity of a tile at a particular site depends on tiles arbitrarily far away from its immediate neighborhood. It seems unlikely that a physical growth mechanism can enforce these dependencies at all.

nonperiodic structures require the use of units either more geometrically precise, or more tolerant of strain than periodic structures.[15]

(iii) Tilings are abstract, infinite objects that can use an arbitrary number of tile types and spring into existence the moment we imagine them; in contrast, real crystals are finite, the number of tile types we can reasonably create is small, and the amount of time we can wait for them to assemble is bounded. Where, in the mathematical theory of tiling, finding a minimal set of tiles with certain properties is a game, in the real world finding the minimal set of tiles with certain properties may dictate whether a desired structure can be self-assembled. Thus it is important to use complexity theory to study the program-size complexity (size of a tile set) and time-complexity for self-assembly.

While difficult, these challenges do not seem insurmountable. Winfree, with Nadrian Seeman's group, has taken the first step: they designed and then experimentally verified periodic two-dimensional DNA lattices containing up to four different DNA tiles [WLWS98b][16]. But these simple structures, like a Turing machine that ignores its input and outputs an infinite string of "0"s, can be said to perform computation only in the

[15]The size that a growing crystal can attain is dictated, in part, by how well its basic units fit together geometrically. It seems unlikely that we can make all of a set of tiles exactly the same size and shape. Periodic lattices often have symmetries that partially distribute the strain associated with such geometric mismatches. In general, tilings that simulate Turing machines have no such symmetries. For example, an arbitrary number of tiles of type A may appear in a row over the same number of a different type of tile, say B. If the size of A is a little larger than the size of B, say $|A| = |B| + \epsilon$, then a row of n As adjacent to a row of n Bs will have a length mismatch of $n\epsilon$. Thus nonperiodic tilings may accumulate small size discrepancies until they become large ones. If the tiling is grown by adding As on top of Bs then as the number of sequential As approaches $|B|/\epsilon$ something must give — an A tile may be excluded or may bind in an incorrect orientation forming a defect that may hinder further growth. This phenomenon, the problem of "mismatched lattice constants" for dissimilar units, is a major challenge in the growth of solid state inorganic materials. Since DNA can accommodate some amount of over- and underwinding, we hope that it can absorb the strain from such mismatches when they occur in nonperiodic structures.

[16]Since, Mao *et al.* have explored algorithmic assemblies intermediate in computational power between linear double-helical assemblies and two dimensional assemblies; they have created linear structures with three double helices that perform the cumulative XOR of two binary numbers [MLRS00].

most trivial sense. The experiments performed so far do not address the frequency of errors, control of nucleation, or other difficulties associated with tilings designed to simulate arbitrary Turing machines. Much work remains to be done before we can say that we have truly made algorithmic self-assemblies.

1.3 An Example Self-Assembled Computation

To understand the challenges involved in embedding computations in self-assembly and the questions that unify this thesis, we now discuss a simple example. Consider the seven objects pictured in Figure 1.1a (from [Win00]). These are the aforementioned Wang tiles (from now on, just tiles) but we intend that they be interpreted as any objects "geometrically equivalent"[17] to squares, and bearing up to four different binding domains. Together these tiles constitute a *self-assembly program* for counting in binary, hence we call them *counter tiles*. Conceptually, we divide the tiles further. Tiles bearing large letters are known as *nucleating tiles* — they set up the initial conditions or input for the computation. Tiles bearing large numbers are known as *rule tiles*; they perform the computation and their numbers are to be interpreted as binary digits. Small labels on the edges of the tiles represent different kinds of binding interactions. Lines on the edges indicate the strength of binding: thin single lines indicate strength-1 (in some unit of energy) bonds, thin double lines indicate strength-2 bonds, and thick lines indicate strength-0 interactions.

[17] By this we mean that they are topologically equivalent to squares (*i.e.* they are topologically equivalent to disks) and for all assemblies of the objects, a graph indicating the bonds in an assembly is isomorphic to a subgraph of the square grid. Penrose tiles, which we study in Chapter 3 are an exception to this; the adjacency graph of a Penrose tiling is 4-connected but is not isomorphic to any subgraph of the square grid.

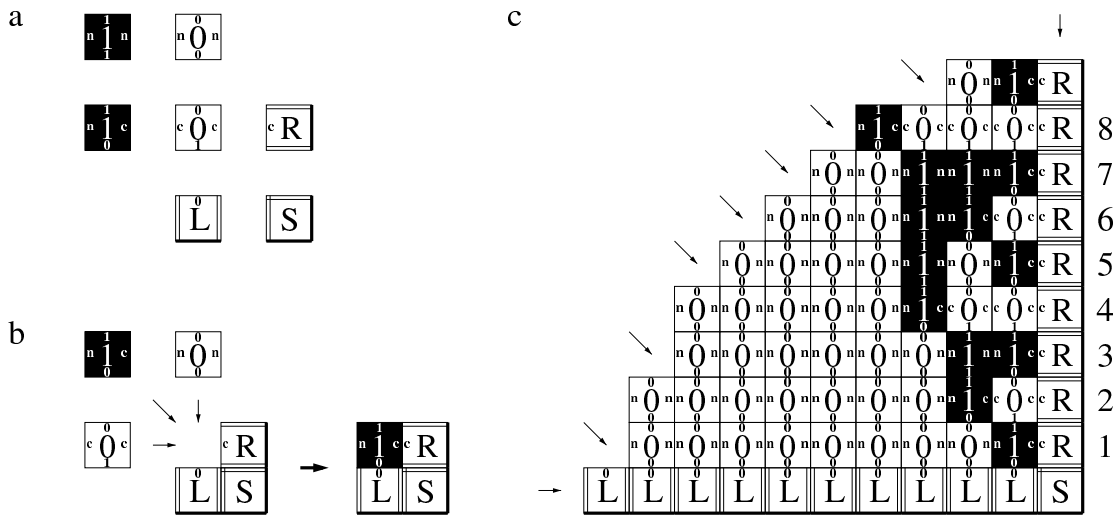


Figure 1.1: A set of tiles that count in binary.

Consider the combination of counter tiles in a world with a simplified physics. (In Chapter 2 we use a formal model to capture this simplified physics, Erik Winfree's Tile Assembly model.) Imagine that each type of tile is available in unlimited numbers. Let us make the idealization that a particular tile can stick to another tile or aggregate of tiles only if the sum of the strengths of the bonds between them is $\geq \mathcal{T}$, where $\mathcal{T}$ is a single parameter capturing the "experimental conditions" in our ideal world — it is a function of absolute temperature, tile concentration, and bond strengths. Consider first that $\mathcal{T} = 2$; then tiles can only bind to other tiles if they bind with a total strength ≥ 2. A corollary of this assumption is that the rule tiles, which can form only strength-1 bonds, can only bind to an aggregate if two or more bonds cooperate to hold the tile in place. Thus, at first, the only counter tiles which can assemble are nucleating tiles — via strength-2 bonds. For example, an "L" tile may bond on either side to another "L" tile or on its right side to an "S" tile. Only after nucleating tiles have begun to assemble into an "L-shape", can rule

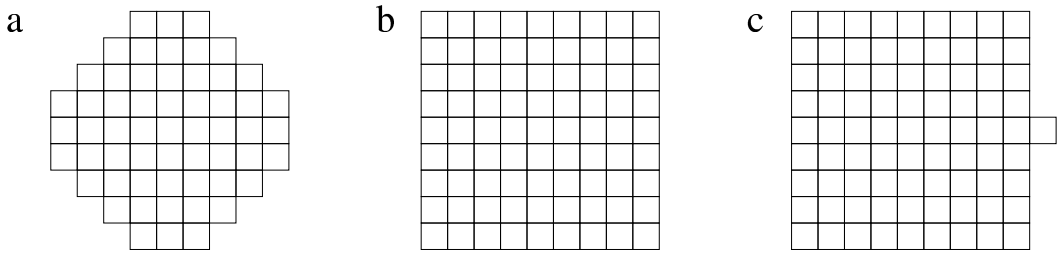


Figure 1.2: Real crystal growth approximates $\mathcal{T} = 2$ assembly.

tiles begin binding at *corner sites* as shown in Figure 1.1b. Here we have shown three rule tiles, all which match the corner site at one or more sides; only one of these tiles can form two strength-1 bonds and so is the only one that sticks.

Successive additions of rule tiles and nucleating tiles would result in a structure like that in Figure 1.1c whose rows may be read, from bottom to top, as an enumeration of binary numbers. To understand how this works, inspect the rule tiles. Consider the bottom and right sides of each rule tile as *inputs*, and the left and top sides as *outputs*. A rule tile fitting into a corner "reads" two input bits by matching bonds; one bit it reads is the identity of the digit below it and the other is a carry bit from the tile to its right (if "c", carry= 1; if "n", carry= 0). The number on the rule tile and the bond that it outputs on its top reflect the result of adding, modulo 2, the two input bits; the bond it outputs to its left reflects the resulting carry bit. Rule tiles thus copy the digits below them, unless a carry is indicated from the right. Initially the "L" nucleating tiles present all zeros to the rule tiles from below; this starts the counting at zero. The "R" nucleating tiles present a new carry bit for each row of the counter from the right; this adds 1 to each successive row of the counter.

It is clear from Figure 1.1c that multiple corner sites may be available for binding rule tiles at the same time; the order in which tiles are added at these sites is not specified. Despite the nondeterministic nature of assembly, it can be shown that the infinite structure that is formed by the counter tiles is unique [Win98a]. Informally[18], this is because a unique rule tile binds at each corner site. For the counter tiles, this in turn is a consequence of our assumption that a rule tile maybe added only by the cooperative formation of at least two bonds at once, that is, $\mathcal{T} = 2$.

Now consider the alternative, that $\mathcal{T} = 1$. Under such conditions, rule tiles can bind to each other, via strength-1 bonds, without the aid of nucleating tiles to initiate the structure. And, at each corner site, as depicted in Figure 1.1b, two tiles different from the desired tile could bind; the mismatch between tiles created by such an event is termed an error. Thus the number of undesired structures consistent with $\mathcal{T} = 1$ assembly is enormous; informally, the probability of assembling a binary counter of any reasonable size is very small. Even starting from a correctly formed "L" of nucleating tiles, the number of incorrect structures that could form is exponential in the number of tiles added. Thus, in the simplified physics we have presented, our assumption that $\mathcal{T} = 2$ is necessary for correct assembly of the counter tiles.

Experimentally, $\mathcal{T} = 2$ conditions appear to be relatively difficult conditions to achieve — we must carefully tune temperature, concentration and bond strengths to maintain $\mathcal{T} = 2$. Further it appears that self-assembly proceeds slowly under $\mathcal{T} = 2$ conditions

[18] In Chapter 2 we determine conditions for the uniqueness of finite self-assembled structures for the Tile Assembly Model, and give one heuristic for designing them.

[Win98a]. $\mathcal{T} = 1$ conditions, on the other hand, seem easier to implement — concentrations, for example can be as high as allowed by solubility. Further, we are free to pick temperatures and concentrations that optimize the speed of assembly. Given the difficulties associated with $\mathcal{T} = 2$, how important is $\mathcal{T} = 2$ assembly? Put another way, if $\mathcal{T} = 2$ appears more difficult to maintain experimentally, why not use $\mathcal{T} = 1$?

We have no rigorous answers to these questions but our intuition and experience points to $\mathcal{T} = 2$ assembly as more interesting and promising than $\mathcal{T} = 1$ assembly. First, Winfree has shown that, under the Tile Assembly Model, $\mathcal{T} = 2$ assembly is universal — in particular that one-dimensional cellular automata can be simulated using 2D self-assembly. His construction *requires* $\mathcal{T} = 2$ because it makes decisions based on pairs of bonds (as did the counter above). Further it has a number of nice properties: (i) it is relatively fast since computation can occur in parallel along the edge of a structure, (ii) for many interesting patterns (like the Sierpinski triangle we'll meet in the next section) it uses only a few types of tiles, and (iii) it can be programmed to generate a unique structure of interest (without wasting tiles in other undesired structures).

While $\mathcal{T} = 1$ assembly is capable of universal computation, the only known construction (due to Len Adleman[19]) simulates Turing machines and operates in a wholly different manner from Winfree's $\mathcal{T} = 2$ construction for cellular automata[20]. Rather than depend on the creation of pairs of bonds to make decisions, it relies on geometry: essentially one dimensional in nature, structures wind around the plane running into themselves (or not) depending on the state of the computation. The construction's clever use of geometry is

[19]The $\mathcal{T} = 1$ construction to which we allude was presented by Len to the USC self-assembly theory group. We discuss the construction and related issues a little more in Chapter 2 Section 2.6.5.

[20]Adleman's construction is also very different from $\mathcal{T} = 2$ constructions for simulating Turing machines like the construction we give in Chapter 2 on p. 111.

what makes it very interesting; unfortunately, we do not have a molecular self-assembly system with rigid enough units to implement this scheme. Perhaps more importantly, the construction computes slowly, uses more tiles, creates dead-end structures[21] and does not appear to be capable of forming the patterns available to $\mathcal{T} = 2$ constructions. Whether or not $\mathcal{T} = 2$ is strictly necessary for efficient universal computation, all of the computational examples we know from the literature of tilings require $\mathcal{T} = 2$ assembly (*i.e.* computing primes [GS87]), as do all of the non-trivial constructions in this thesis.

The concept that we might be able to compute by relying on $\mathcal{T} = 2$ growth is not such a wild one. The self-assembly of crystals (or, at least, the growth of good crystals) is a process that approximates $\mathcal{T} = 2$ growth. Otherwise crystals would not take on their characteristic faceted shape. For example, consider the cartoon crystal in Figure 1.2a; tiles add preferentially on its diagonal faces to fill out the square shape of the crystal in Figure 1.2b (see [Lau70] pp. 83–86 for a discussion of this using similar figures). At the same time we know that crystal growth cannot be exactly $\mathcal{T} = 2$ or crystals (which have no equivalent of nucleating tiles) would never nucleate and grow. For example, the crystal in Figure 1.2b could never be extended as in Figure 1.2c. In fact, more physical models for crystal growth generalize $\mathcal{T}$ as a continuous parameter and make tile additions probabilistically. Using such a model, Winfree has characterized the tradeoff between speed of growth and error rate for different values of $\mathcal{T}$ between 0 and 2 [Win98a]. Importantly, he has found that low error rates come at the price of speed — error free crystal growth appears under conditions arbitrarily close to $\mathcal{T} = 2$ and takes an arbitrarily long time. Under reasonable assumptions for DNA self-assembly, he has found that 200×200 tile

[21] At least in 2D it does. See Section 2.6.5.

aggregates grown over the course of a week will contain, on average, about a single error. Error correction schemes, which have not been extensively explored, may allow longer self-assembled computations to be performed at a given error rate.

1.4 Teleology: *But what is it good for?*

If an experimental system modeled by $\mathcal{T} = 2$ can be found, one can use two-dimensional self-assembly to build a binary counter. And, subject to the same caveats, one can perform universal computation. That is, any computer program may be translated into a set of tiles that when self-assembled, simulate the computer program. This is intriguing and suggests that any complete description of self-assembly must be powerful enough to describe universal computation. It implies that certain questions about the physical world are undecidable: for example, given an arbitrary set of molecular building blocks and rules for their self-assembly, it is impossible to write a computer program to decide whether (i) the molecules can only be self-assembled to form structures of finite size, or (ii) the molecules can be self-assembled to form structures of an arbitrary size.[22] But the stubbornly practical (non-computer scientist) may still ask: *What is such an embedding of computation in self-assembly good for?*

In principle we could use self-assembly wherever we use a conventional computer; in practice we do not expect that computation by self-assembly will be able to compete with traditional computer architectures on explicitly computational problems[23]. Thus, instead,

[22] This is a point often raised by Len Adleman, also raised by Warren Smith [Smi97], and follows from the undecidability of the Halting Problem.

[23] See Chapter 5 for a comparison of the performance of electronic computers to one model of DNA computation on a specific computational problem. While the model of DNA computation studied is quite different from self-assembly, we expect the analysis for DNA self-assembly to be similar.

we are motivated to look for *fabrication problems*: particular patterns or sets of patterns that we expect will have useful properties (*e.g.* as templates for electronic circuits) and that may be accessible by self-assembly. This point of view stimulates us to ask how computation can be used for fabrication problems and, more generally what implications universal computation has for the patterns that can be formed by self-assembly.

One approach[24] to these very broad questions is to ask a more concrete question and see where it leads us. For example, "Can we self-assemble a Pentium?" Assuming that we can create tiles that act as circuit elements[25] what we are really asking is "Can we self-assemble the pattern underlying a Pentium?" The answer, it turns out, is yes and we may do so without using any complex computation. First, by *pattern* we mean the coloring of an arrangement of tiles independent of the underlying tile types. Then any particular pattern, no matter how complex, can be self-assembled by assigning a unique tile type, with a unique set of binding interactions with its neighbors, to each position in the pattern and painting it the appropriate color for its position. But this type of

[24]Another approach, tempting to the computer scientist, is to conclude that "achieving universal computation in self-assembly means that the sets of patterns that can be created by self-assembly are as broad as can be imagined, arbitrarily complex". Besides its imprecision, such a statement is flawed for at least two other reasons. First, although it sounds nice and makes the connection between computation and self-assembly sound important, it does not really tell us anything. Decoded, the statement is a tautology; arbitrarily complex sets of patterns are, according to Church's thesis, exactly those sets of patterns that can be computed by a universal computer. Really, for the statement to be impressive to someone, that someone has to be familiar with the patterns that computer scientists have learned we can create with computers and be impressed by those patterns. As a computer scientist familiar (somewhat) with the patterns that computers can create I still have little insight into which, if any, of these patterns will be useful for self-assembly. A second flaw in the above statement is that, for 2D patterns created by 2D self-assembly, it is not true! As we will see in Chapter 2 the sets of patterns that tiles can form are actually decidable, and thus cannot be "arbitrarily complex". If we insist on using 2D assembly to make 2D patterns then only by augmenting self-assembly with other special operations such as the ability to *destroy* a subset of tile types *after* self-assembly can we create all computable sets of 2D patterns. If we are willing to use 3D self-assembly then we can generate all computable sets of 2D patterns as the faces of 3D structures (as we can make all computable sets of 1D patterns on the edge of a 2D structure that simulates a Turing machine). But then, without a destroy operation, our 2D sets have awkward 3D objects attached to them that encode the entire history of their computation.

[25]Periodic electrical networks of functional LEDs have already been self-assembled on the millimeter scale [GTB+00].

self-assembly program, which we call *uniquely addressed*, is undesirable because it is not *tile-efficient* — a tile-efficient program (defined in Chapter 2) would use a small number of tile types compared to the size of the pattern. Instead unique addressing uses the greatest number of tile types possible to create a pattern. In a physical implementation it appears that creating unique tile types and unique binding interactions (that are specific) is expensive and difficult — it seems that unique addressing is impractical.

Given that we are not interested in self-assembling a Pentium by using a unique tile for every transistor we refine our question again: "Can we self-assemble, by using computation rather than unique addressing, the pattern underlying a Pentium tile-efficiently?" We conjecture that the answer is no — we believe that a Pentium has too much random structure (that would require unique addressing) to be self-assembled tile-efficiently. If true, the pattern for a Pentium is not alone. For almost all patterns, by results in Kolmogorov complexity[26], unique addressing is provably the best one can do. Thus even though an arbitrary pattern *may* be self-assembled, for almost all patterns self-assembly is probably impractical (or no more practical than direct lithographic methods.)

Does the last result mean that using algorithmic self-assembly to create patterns is hopeless? Fortunately not: similar complexity results place similar limits on computer programs yet in practice we continue to write small computer programs for interesting patterns — for example the complicated 32×32 matrices in Figure 1.6 were generated by a computer program of only $O(\log \log 32)$ bits in size. The last result does mean that our concrete question should be broadened to: "What interesting patterns *can* be tile-efficiently

[26] See Chapter 2 for a definition of *almost all* and a explanation of why this is so.

generated by algorithmic self-assembly?"[27] This is a difficult theoretical question, one for which we have little insight, and our experience with computer programs is not always of help. Consider the example of a circle — it is easy to write a small computer program that can draw pixel-based approximations of circles but we have no idea how to specify a tile-efficient self-assembly program that yield circles. We have equally little insight into the dual problem: given a pattern of interest, how to best use computational self-assembly to construct an approximation of it. Given the theoretical difficulty of these problems we expect little in the way of general solutions but, as is true for computer programs, we expect good solutions for specific classes of patterns.

What we have so far are a couple of examples for which simple computations, using a "small" tile-efficient number of tile types, may allow the construction of interesting physical patterns. One example is the set of counter tiles, depicted in Figure 1.1. The counter tiles use a constant number of tile types to form a structure that grows indefinitely in two directions. By augmenting these tiles with just $O(\log N)$ tiles we can control the growth to create a rectangle of exactly size $N \times \log N$. Thus the counter can be used to make relatively narrow structures of a chosen length. By adding an additional constant number of tiles we can self assemble $N \times N$ squares. (In Chapter 2 we show the details.) Thus very large structures, of exactly size $N \times N$, can be made from a small number of types of tiles — $O(\log N)$. In these examples, wedding computation with self-assembly addresses what is to chemists a difficult synthetic (fabrication) problem — how to make polymer or crystalline structures of a well-defined size.

[27] Put another way, we are asking what patterns can be efficiently compressed using self-assembly programs.

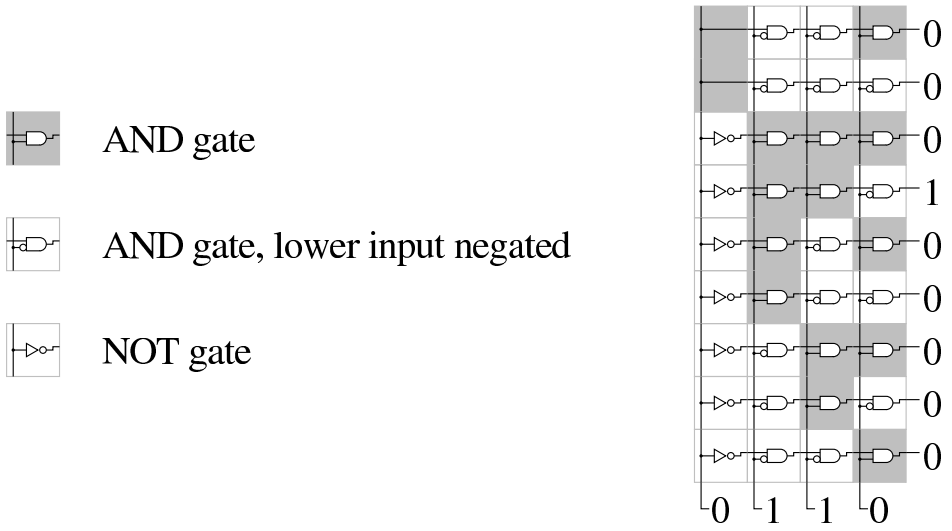


Figure 1.3: Using a binary counter to self-assemble a demultiplexer.

Perhaps surprisingly, the binary counter also has a completely different physical interpretation, one that is equally interesting. The counter tiles can also be used for creating a self-assembled circuit. Pictured in Figure 1.3 is a fixed width binary counter, four bits wide. Tiles shaded gray indicate tiles labeled "1" in Figure 1.1 and tiles shaded white indicate tiles labeled "0" in Figure 1.1. In this version of the counter, tiles representing most significant bits of the counter, along its left edge, are distinct from tiles internal to the counter. Before self-assembly[28] tiles are decorated with logic gates and wires in the following way: Most significant bit (MSB) tiles representing "1" have an input wire that enters on the bottom of the tile and splits into two outputs, one on the top and one on the right. An MSB tile representing "0" is like a "1" MSB tile except its right output is negated by a NOT gate. Internal "1" tiles have an input on the bottom that splits into an output on the top and a wire that serves as one input for an AND gate. The other input

[28] Or after if the logic gates can be made to stick to the appropriate types of tiles.

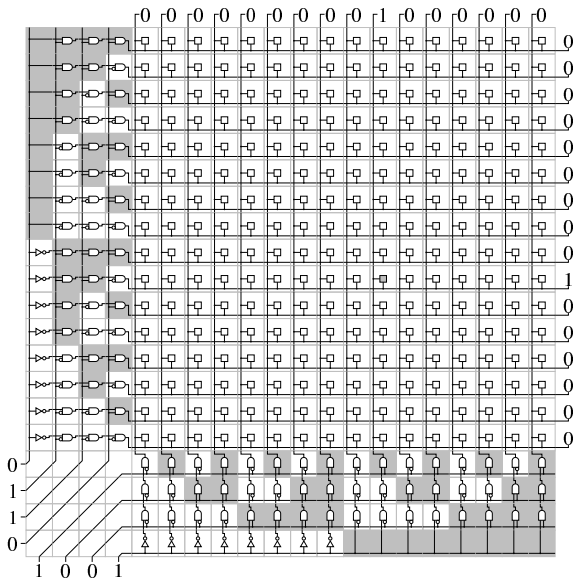


Figure 1.4: Two self-assembled demultiplexers at right angles can address a memory. The gray memory cell is being addressed in this figure.

from the AND gate enters the tile from the left and the output of the AND gate leaves the right side of the tile. An internal "0" tile is like an internal "1" tile except the input from the bottom is negated before the AND gate. Once assembled, the tiles form a circuit with 4 input lines along the bottom and $2^4 = 16$ output lines along the right.

If the four input lines are driven with the bits of the string "0110", copies of these inputs propagate from bottom to top along the columns of tiles. In the kth row of the circuit the input bits or their negations are ANDed together according to the binary digits of k. If and only if the binary digits of k match the input bits does the value of the kth output equal one; for "0110" only the sixth output line equals one. Thus the circuit self-assembled by the counter tiles is a demultiplexer; a generalized version can take an n-bit input and turn on one of 2^n output lines.

A demultiplexer is an interesting, and perhaps useful, self-assembled circuit; a pair of demultiplexers oriented at right angles along the borders of an $N \times N$ memory allow a memory element to be accessed using only $2 \log N$ lines — thus a memory circuit may be self-assembled (see Figure 1.4). What other circuits might be possible? If we inspect the demultiplexer circuit a little more abstractly we find that what it really computes is the multiplication of a binary vector by a binary matrix, with the binary function EQUIVALENCE substituting for multiplication and AND substituting for addition in the definition of matrix multiplication. The circuit takes an n-bit binary vector, multiplies it by a $n \times 2^n$ size binary "counting" matrix, and outputs a 2^n long vector. Similarly, a circuit for an arbitrary binary matrix multiplication could be created by self-assembling a circuit decorated with logic gates as described for the binary counter. Decorations with different logic gates could implement different definitions of multiplication. What matrices and matrix multiplications (besides the binary counting matrix and multiplication defined) yield interesting circuits is an open question; later we give a couple candidates.

Another complex pattern that may be created by a simple self-assembling computation is the Sierpinski triangle, pictured in Figure 1.5a. Only seven tiles (Figure 1.5b) (from [Win98a]) are required to create a discrete approximation (Figure 1.5c) of this triangular fractal pattern and, as for the counter tiles, its construction depends on $\mathcal{T} = 2$ assembly. By labeling the sides of the tiles as "input" and "output" as described in Section 1.3 individual tiles can be seen to encode the binary function XOR. Diagonals of the assembly, interpreted as zeros and ones, form rows of Pascal's triangle modulo 2. It can also be seen that diagonals of the assembly are instantaneous descriptions of a one-dimensional cellular automaton. Aside from its interpretation as a computation, this pattern is beginning to

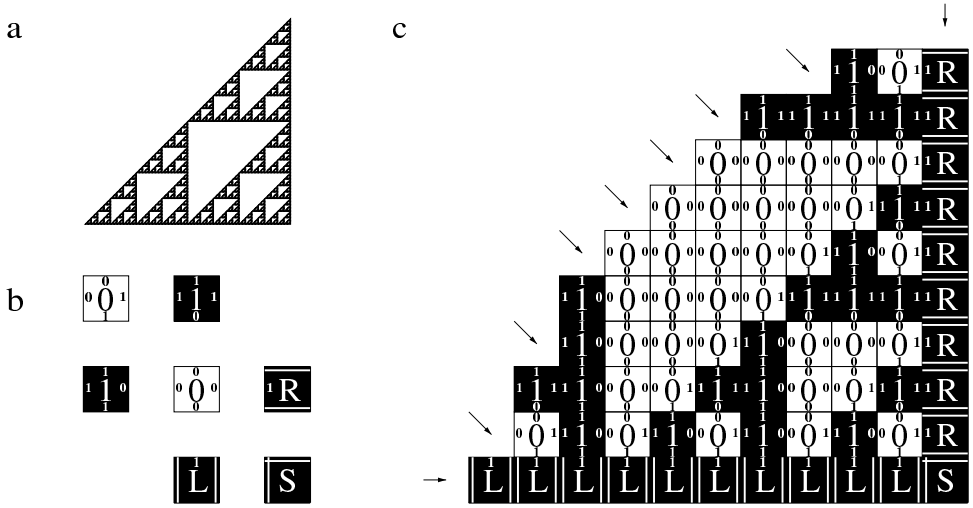


Figure 1.5: The Sierpinksi triangle and a set of tiles that approximates it.

find some practical uses; rendered in metal the Sierpinski triangle appears to be a superior cellular phone antenna [PBRPC98].

Does the Sierpinski triangle also have a circuit interpretation like the binary counter? Perhaps not, but it inspires thought: interpreted as a binary matrix the Sierpinski triangle has many periodic rows whose periods are related by a logarithmic scaling. This suggests that using the Sierpinski triangle as a matrix multiplier might effect some transform similar to a wavelet or Fourier transform. In fact the binary versions of the wavelet and Fourier transforms, a binary pseudowavelet transform [PB99] and the Hadamard transform [PKA69, YH97], have self-similar matrices closely related to the Sierpinski triangle (See Figure 1.6 and Appendix A). Both these transforms are useful in signal processing and image compression; the Hadamard transform in particular, has uses from quantum computation to cell phones. Given the similarity of these transforms to the Sierpinski triangle, it may be possible to self-assemble circuits to compute them.

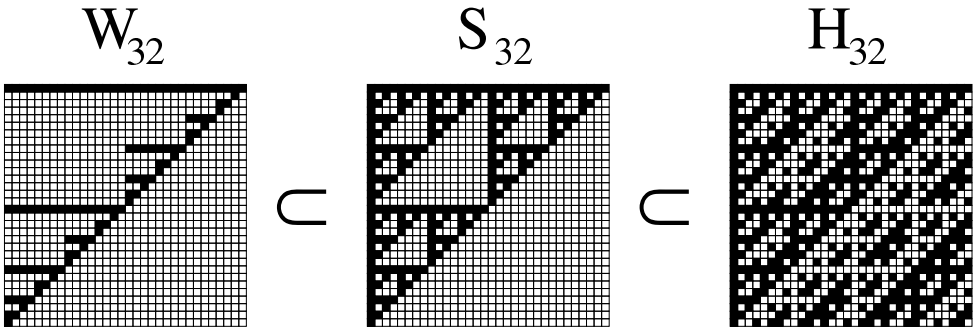


Figure 1.6: Comparison of self-similar binary matrices. From left to right, the binary pseudowavelet matrix (W_{32}), the Sierpinski triangle matrix (S_{32}), and the Hadamard matrix (H_{32}). For the pseudowavelet and Sierpinski matrices, black represents 1 and white represents 0. For the Hadamard matrix, black represents 1 and white represents -1. These matrices, W_n,S_n,and H_n are defined for size $n = 2^k$, $k \geq 1$. For matrices A and B we say that $A \subset B$ iff A_{ij} is black $\implies$ B_{ij} is black and $A \neq B$. For all $k \geq 2$ $W_n \subset S_n \subset H_n$.

While none of the above examples seem as impressive as a self-assembled car or self-assembled Pentium, they give us hope that by adding a little computation to the process of self-assembly we can create structures much more complicated than the blandly periodic structures created by self-assembly so far. It must be emphasized that the examples above in no way completely characterize the kinds of patterns we can self-assemble using computation. Poetically, they are just a couple of seashells found at the edge of an ocean of possibility and we are in danger of searching too close to shore: attempting to generate interesting self-assembly programs by looking for ways to shoehorn known circuits or Turing machines onto self-assembled patterns unnecessarily limits our exploration of algorithmic self-assembly. Even thinking about self-assembly in terms of cellular automata, which seems quite a natural thing to do, may hinder our imagination. Thus, we should not settle

on one computational model for self-assembly or one way of programming self-assembly too quickly.[29]

In many ways, our ignorance of self-assembly parallels our ignorance of neural networks; neural networks are formally equivalent to all other general purpose computers and we have an existence proof (our brains) that computers based on neurons can perform amazing computations — yet we have relatively few examples of neural networks that do useful things and we really have no idea how best to program them. We are similarly inspired by natural self-assembly — and similarly ignorant of the patterns that can be built by self-assembly. This profound ignorance creates the possibility of great discovery and makes the present an exciting time to study algorithmic self-assembly.

1.5 Organization of this Thesis

In this thesis we explore, first theoretically and then experimentally, the dichotomy between $\mathcal{T} = 1$ and $\mathcal{T} = 2$ assembly. While universal computation can be performed by self-assembly under both conditions, it appears that computation can be used to greater effect at $\mathcal{T} = 2$. This apparent difference seems to underlie a provable difference in program-size complexity for $\mathcal{T} = 1$ and $\mathcal{T} = 2$ assembly: In Chapter 2 we study the program-size complexity of squares — the minimum number of tile types required to self-assemble an $N \times N$ square — and find that using computation under $\mathcal{T} = 2$ gives a tile-efficient $O(\log N)$ complexity whereas under $\mathcal{T} = 1$ computation cannot help; the complexity is N^2. If $\mathcal{T} = 2$ assembly can be realized experimentally, this result has practical implications for

[29] We note that even for "classical" von Neumann architectures there is no single best way to program: functional, procedural, logic-based, and object-oriented programming languages and methods all have their supporters.

the self-assembly of objects of defined size. In addition, we give methods for proving our self-assembly programs correct, outline what we think are appropriate notions for efficient self-assembly programs, and review the status of various complexity theoretic questions for self-assembly programs.

Self-assembly in real systems likely lies between $\mathcal{T} = 1$ and $\mathcal{T} = 2$; thus finding systems and experimental conditions that approximate $\mathcal{T} = 2$ is an important goal. To this end, in Chapter 3, we explore an experimental system in which plastic tiles self-assemble via capillary forces. First, we test our ability to use the system to create specific binding domains: we explore the assembly of a set of tiles designed to form a simple periodic "checkerboard" pattern. Next we explore the assembly of a more complex set of tiles designed to form the aperiodic Penrose tiling. Finally, we test the system's ability to compute using a set of "XOR" tiles that, if nucleated appropriately under $\mathcal{T} = 2$, would simulate a one-dimensional cellular automaton to form a Sierpinski triangle but, under $\mathcal{T} = 1$, would assemble into "meaningless" aggregates full of errors. Our experiments yield intermediate results: the XOR tiles spontaneously nucleate to form aggregates that simulate the desired cellular automata on a random input with few errors. Unfortunately, the growth mechanism we observe appears incompatible with the nucleation of computations to use a chosen input (*e.g.* the particular input that would yield a Sierpinski triangle).

In Chapter 4 we describe a DNA-based system for exploring the phenomena of errors and nucleation. We design DNA tiles to generate two-dimensional "bar-code" lattices bearing a random pattern of stripes; correct $\mathcal{T} = 2$ growth of the lattice would copy the pattern of stripes indefinitely. An error results in the conversion of a stripe to a non-stripe, or vice versa, a feature we find readily visible under the atomic force microscope,

allowing future studies to optimize conditions for $\mathcal{T} = 2$ growth. With DNA melting studies we learn that spontaneous nucleation of the bar code lattices can be delayed up to 500 minutes. So far, attempts to nucleate bar-code lattices on a chosen pattern of stripes have been unsuccessful.

Finally, in Chapter 5, we give an algorithm for the cryptanalysis of the Data Encryption Standard (DES) using the Sticker Model of molecular computation [RWB+98]. We give an estimate for the amount of time required to break DES based on a a detailed accounting of operations in the molecular algorithm. We estimate the space (in terms of mass of DNA) required to break DES based on a very simple error model. While not about algorithmic self-assembly, this chapter is related to the other chapters since it was written as part of a broader endeavor to apply ideas from computer science to the biological and chemical sciences.

1.6 Sources, Collaborators and Comments

The experimental work presented in this thesis was all performed by me, all of the figures and writing are mine except where noted below. I am solely responsible for any mistakes herein. Please feel free to write me with corrections, comments or questions at pwkr@alumni.caltech.edu.

In this chapter, the constructions for the binary counter and Sierpinski triangle are due to Erik Winfree, as is the idea to decorate the binary counter with circuit elements to form a demultiplexer. The idea of viewing self-assembled patterns as matrix transforms is due to me, as is the application to binary pseudowavelet matrices and Hadamard matrices (Appendix A). Matt Cook and I have been working on tile sets to create these patterns.

Chapter 2 draws heavily from [RW00] which appeared in the proceedings of *STOC '00* represents a joint effort between Erik Winfree and me. Len Adleman provided the original motivation for this work by posing the question "What is the complexity of generating an $N \times N$ square by self-assembly?" Erik Winfree was the sole author of the original draft of this paper which included the clever counting constructions for forming squares with $O(\log N)$ tiles, the construction that for some N, squares could be formed with $O(\log^* N)$ tiles, as well as the link to Kolmogorov complexity. I provided the construction that links the assembly of squares to Turing machines. The rest of the square constructions we either both generated independently, or generated together. Len Adleman originally guided the formalization of the model of self-assembly. The algorithm for determining whether a set of tiles uniquely produces a given assembly is due to David Kempe and me. The proof of this algorithm given here, as well as the algorithm and proofs for exact production and unique production of *ti*-assemblies are due to me. Matt Cook's detailed comments helped simplify the proofs and streamline their presentation; several errors would have gone unnoticed without him.

Chapter 3 appeared as a featured article in Proceedings of the National Academy of Sciences [Rot00]. One of my pictures of Penrose tiles appeared on the cover; the same picture appears at the end of this thesis on page 283. The research was inspired by two sources: Ned Bowden and George Whitesides' system for lateral capillary force self-assembly and Erik Winfree's idea that two-dimensional self-assembly could be used to compute — the idea to combine these two systems is my own. The idea to use hydrophobic sand to visualize menisci is due to me, as are the wetting codes that decorate the tiles. Len Adleman provided many suggestions, heavily influenced the direction of the research

and helped me extensively with rewriting the paper. John Lyons of Laser Connection in Arcadia, CA laser cut all of the tiles from my original designs. K. Bui and Karen Casciotti helped me protect some of the tiles with tape.

The text of the PNAS article appears here with little change from its original form — I have made a few things clearer, corrected two errors in the references, and added a couple figures, footnote, and a section on growth rules for Penrose tile self-assembly. The most important addition is a single equation which, trivial though it may be, correctly predicts a transition in the behavior of the tiles. To help the reader understand the physical basis of the capillary forces and surface tension, which is often poorly explained, I include a brief explanation of them in Appendix B. Also, because the vertex star statistics of self-assembled Penrose tiles seemed different than those of mathematical Penrose tilings, I computed the statistics for mathematical Penrose tilings. While the answer merits one line in the PNAS paper, the possibility of saving someone else the calculation causes me to include the details in Appendix C. The idea that capillary forces be used to make low friction gears is due to Len Adleman and me — preliminary efforts in this direction appear in Appendix D. Only peripherally related to the rest of the thesis, it speculates a little on possibilities for the self-assembly of such gears and hints that lateral capillary forces may have more interesting behavior, yet to be explored.

Chapter 4 is really a work in progress and represents a collaboration between Len Adleman, Erik Winfree and me. The computer program used to generate sequences for the bar code tiles was written by Erik Winfree. John Petruska pointed out that the noise in our UV melt data was due to light scattering off of large crystals. The idea to use *in situ*

PCR to effect annealing of the tiles is due to Nickolas Chelyapov. Bernie Yurke pointed out the problems inherent in linear self-assembly with incorrect stoichiometry.

Chapter 5 appeared in *The Journal of Computational Biology* as joint work with Len Adleman, Sam Roweis, and Erik Winfree [ARRW99]. Len Adleman wrote the original draft of this paper, and I rewrote it extensively. Most of the intellectual work behind the paper is Len's; it is my observation that distractors are a constant fraction of the DNA input to a DES computation. Sam and Erik made sure the paper was correct and fit for general consumption. I have added a coda to the end to present my current views on the significance of this chapter and DNA computation in general.

Chapter 2

The Program-size Complexity of Self-assembled Squares

How complex could something as simple as a square be?

—naive graduate student.

2.1 Introduction

To motivate our study of the program-size complexity of self-assembled squares[1], we first consider the importance of complexity measures in the more general context of chemical synthesis. When a chemist devises a plan for the synthesis of a complicated organic molecule, she uses a *divide and conquer* strategy known as *retrosynthesis*.[2] In her mind, the chemist decomposes the complex molecule into two or more parts that can be joined according to a chemical reaction that she knows. Then the chemist selects each of the parts in succession, and decomposes each one over other chemical reactions that she knows. The process continues recursively until all molecules left are cheaply and readily commercially

[1]This chapter represents joint work with Erik Winfree (first published in [RW00] and David Kempe (unpublished). See Chapter 1 Section 1.6 for details.

[2]Retrosynthesis was formalized in 1967 by E.J. Corey, who, in 1990, received a Nobel Prize for his efforts. Note to computer scientists: export the right algorithmic idea to chemistry and win a Nobel Prize!

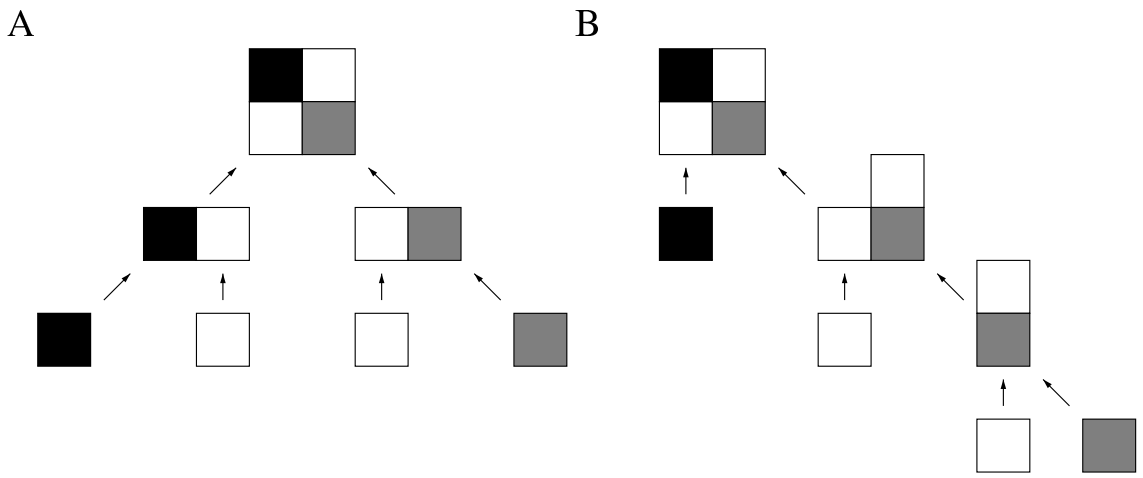


Figure 2.1: Two different synthetic trees (parses) for a single molecule. Molecules are represented schematically by one or more shaded squares. The root of each a tree represents a molecule of interest. Leaves represent starting materials. Otherwise a node represents the product of a chemical reaction involving its children. Tree (A) represents a hierarchical or *convergent* synthesis. The longest synthetic path is two steps. Tree (B) represents a *linear* synthesis. It requires four synthetic steps. Both of these trees have three distinct types of leaves (indicated by gray level) and thus require three different types of starting materials. Thus the synthetic size, height $\times$ #(leaf types), of tree (A) is $2 \times 3 = 6$ and for (B) is $4 \times 3 = 12$. A chemist might thus prefer (A) over (B). For syntheses carried out by self-assembly, rather than a sequential laboratory steps, both the program size and synthetic size reduce to the number of distinct starting materials — they would be the same, 3, for both tree (A) and (B) — according to these measures a chemist should have no preference. In this chapter, however, we will only consider the self-assembly of molecules that proceed by linear synthesis like tree (B).

available[3] *starting materials*. The result is a *synthesis tree* of chemical reactions (see Figure 2.1), not unlike the parse of a string, that, if performed from the leaves to the root should result in the complex molecule desired.

Further, the chemist is interested only in "practical" syntheses. She knows that each chemical step has a certain *yield* — only a fraction of the molecules are transformed correctly at each step. Assuming that all chemical steps have the same yield, the overall

[3]This is actually a precise notion for many academic research chemists. It means: "can be bought from the chemical supplier Sigma-Aldrich."

yield of her desired molecule will be limited by the length of the longest path in the tree. Thus, being greedy, she seeks to minimize the depth of her synthetic tree. Also, the chemist is stingy — she would like to buy as few kinds of starting materials as possible — thus she seeks to minimize the number of distinct types of leaves on the tree.

To a computer scientist the synthetic tree for a molecule is, at its heart, a *program*. The desire of the chemist to find a tree with minimal depth or number of leaves, is simply a desire to find a program having minimal *complexity*. Just as there are many different complexity measures for computer programs — time, space, and program size are a few — there are many different complexity measures one might define for a synthesis. For example, our greedy, stingy, chemist might define the *synthetic complexity* of a molecule as the minimal height $\times$ #(leaf types) product over all possible syntheses for that molecule.[4] A computer scientist studying synthesis might be interested in the minimum number of bits required to describe a synthesis for a molecule, over all possible syntheses for that molecule — the *program-size complexity* for the molecule.

The synthetic complexity of a molecule, above, is clearly well-defined but that the program-size complexity for a molecule can be well-defined is less clear. It seems reasonable to define the program size of a *particular* synthesis to be the length, in bits, of an experimental protocol that describes it. The protocol must include, in addition to a description of the starting materials and synthetic tree, a description of any purification

[4]No single such complexity is accepted by chemists; different chemists would probably stress different factors in a "synthetic complexity". The typical synthetic organic chemist, in most circumstances, seems more greedy than stingy and would weight height (steps) over the number of leaf types (starting materials). This is, I think, because it is rare to be able to re-use the same molecule very often in a single synthesis. Modern combinatorial chemists, however, try to use a small set of molecules, in a combinatorial fashion, to synthesize molecules with arbitrary features — to "span the space" of desired chemical properties. Thus combinatorial chemists would probably emphasize the number of starting materials over number of steps. Still other chemists might include, among other things, the cost/weight of starting materials, or even their environmental impact.

steps or special apparatus required. The machine for executing such a synthetic program is just a chemist capable of reading the protocol and carrying out the synthesis. Thus, given a standard language for describing a synthesis (just a formalization of the language that chemists normally use), the program size of a synthesis may be well-defined, and hence the program-size complexity of a molecule may be well-defined. Like the synthetic complexity, the program-size complexity has *some* practical significance for our chemist — a synthesis for a molecule with minimum program size takes up the smallest space in her notebook.

Unfortunately, because the simulation of arbitrary computer programs can be embedded in the synthesis of a molecule, the program-size complexity of molecules is, in general, uncomputable.[5] Encouragingly, the phenomenon of uncomputability of program-size complexity for computer programs does not prevent computer scientists from placing useful bounds on program-size complexity or minimizing program size for important cases. In a similar manner, as we demonstrate in this chapter, we can place useful bounds on program size for certain problems in synthesis.

[5]Chemists are well aware of the computational intractability of simply finding a synthesis, period, never mind a one that has minimal program size — the combinatorial explosion of possible parses for a molecule puts finding a synthesis beyond exhaustive search. Thus a master synthetic chemist, like the master chess player, uses a combination of her own experience and pattern recognition (sometimes aided by heuristic computer programs) to find a synthesis. And, echoing the arguments of master chess players, some chemists have argued that because of the exponential nature of the retrosynthetic problem, computers will never be able to beat a chemist at retrosynthesis! (I recall this being argued in an advanced synthetic chemistry class at Caltech.)

While it seems obvious that someone should have proven some phrasing of the retrosynthetic problem NP-complete long ago, I can find no reference before Warren Smith's 1997 technical report, [Smi97]. In it, Smith notes that, given a set $\mathcal{S}$ of additive chemical reactions and a molecule $\mathcal{M}$, the problem of whether or not an N step synthesis for $\mathcal{M}$ over the reactions $\mathcal{S}$ exists is NP-complete. Further, for more general transformations (in particular, reactions that remove atoms) he notes that the question of whether $\mathcal{M}$ can by synthesized using $\mathcal{S}$ at all is undecidable (and hence $\mathcal{M}$'s program-size complexity is uncomputable). These statements say nothing about the difficulty of the retrosynthesis problem for any specific set of chemical reactions, however, and it may be possible to find a set of chemical reactions over which all syntheses are easy. In practice, chemists have yet to do so. [Smi97] also discusses algorithms for finding minimum cost syntheses.

In this chapter we study program-size complexity for an important subcategory of general chemical synthesis, *self-assembly*. Informally, self-assembly encompasses those syntheses that are *autonomous* — to synthesize a compound by self-assembly a chemist just mixes the starting materials together. The order of addition of starting materials does not matter, and no purification or other steps are required. Thus, a synthesis by self-assembly, or *self-assembly program* is particularly easy to describe — one just gives a description of the starting materials; hence the *size* of a self-assembly program is just the size of this description.[6] Interestingly, the omission of a particular synthetic tree in the description of a self-assembly program has some significance: as we'll see later many self-assembly programs are compatible with a set of many different synthetic trees and, in practice, the formation of a molecule may follow any one of these trees.

One of the first questions addressed in any treatment of program-size complexity for computer programs is: "What is the program-size complexity of generating a string exactly N bits long?" Thus, as a test case and starting point for exploration, Leonard Adleman has asked the analogous question for 2D self-assembly, "What is the program-size complexity of self-assembling an $N \times N$ square?"[7] In other words: "What is the smallest number of small starting molecules, of roughly the same size and shape, that, when mixed simultaneously in solution, spontaneously form molecules exactly N starting molecules long, N starting molecules wide, totaling N^2 starting molecules in molecular weight?"

[6] Actually the program size might include some bits for specifying the solvent into which the starting materials are mixed, the temperature at which it is held, etc. We assume, however, that all these complications may be folded into a large enough constant and that they will not affect the asymptotic behavior of the program size.

[7] Adleman has also asked the analogous question for time complexity.

This type of question, more abstract than the question of program-size complexity for a specific molecule, is actually of great practical interest to chemists and materials scientists. The preparation of so-called "ideally monodisperse" molecules, all with exactly the same size rather than a distribution of sizes, is a difficult synthetic problem. It is possible to approach ideal monodispersity only for molecules of small size[8], and often requires tedious purification of the molecules post-synthesis. The preparation of nearly identically sized molecules and particles is economically important — for example, the magnetic particles that coat hard drive surfaces must be as monodisperse as possible to achieve high memory densities.

In this chapter, we answer Adleman's question for Erik Winfree's Tile Assembly Model. The Tile Assembly Model assumes that starting materials can be modeled as rigid square tiles, all of the same size.[9] Thus to describe a self-assembly program, one just describes n distinct types of tiles and the interactions between them. For the sets of tiles that we study, it turns out that the formal program size turns out to be $O(n \log n)$ bits. But, when writing self-assembly programs, the number of tile types n seems easier to manipulate and more natural to report. Also, because no synthetic tree is specified for self-assembly programs, n coincides with synthetic complexity for self-assembly programs — thus n

[8]Interestingly, nature gives us a couple examples where large, nearly ideally monodisperse molecules can be created: linear DNA molecules hundreds of thousands of bases long are copied to exactly the same length (up to a very few occasional deletions or insertions), and viral proteins self-assemble reliably into virus particles of approximately 50,000 daltons in mass.

[9]Further, the Tile Assembly Model only treats reactions of a specific form; in particular, it models the association of a single monomer (small starting molecule) with another monomer or larger molecule as in Figure 2.1B. Hierarchical syntheses, or *convergent* syntheses as they are known in chemistry (see Figure 2.1A), are not considered. Hierarchical self-assembly is common in nature, and is sometimes exploited for artificial systems. Here, we have chosen to study non-hierarchical *pseudo-crystalline self-assembly* since (i) it is the flavor of self-assembly for which we have theoretical insight (ii) it appears easiest to realize experimentally. In fact, the distinctions are not so clean: the DNA tiles described in Section 2.2 and Chapter 4 are themselves self-assembled from multiple strands of DNA in a hierarchical fashion.

coincides roughly with a chemist's idea of how practical a particular self-assembly program will be to implement in lab. For these reasons we say that the program size of a self-assembly program is the number of distinct tiles n and remember that we can always convert it to a more formal measure in bits.

Our approach for minimizing program-size is to embed computation in the self-assembly process. During its execution, the computation lays down a template for the assembly of a square — the length of the computation defines the size of the square. By selecting a simple counter for the computation, we find that, for *all* values of N, one can create an $N \times N$ square with just $O(\log N)$ types of tiles. For special N, using more complicated computations, we find that even fewer types of tiles can be used. In fact, by embedding arbitrary computations in the construction of a square, the ratio of the size of a square and the size of its program can be made arbitrarily large. Along the way we suggest notions for efficiency in self-assembly and discuss techniques for proving the correctness of our self-assembly programs.

2.2 Overview of the Tile Assembly Model

The Tile Assembly Model[10] is a formal model for the self-assembly of rigid square units, known as Wang tiles, constrained to self-assemble on a square lattice[11]. The model is an

[10]The Tile Assembly Model, in a less formal description, appears first in Erik Winfree's thesis [Win98a].

[11]It is straightforward to restrict the Tile Assembly Model to 1D, or to extend it to 3D. The 1D case is uninteresting both in terms of program-size complexity (for a segment N tiles long) and computation. It is easy to see, by a pumping argument, that to produce a 1D line of exactly N tiles requires N distinct types of tiles. Further, the sets of structures that can be self-assembled in 1D can be shown to be equivalent to the regular languages [Win98a]. These results parallel the decidability of the 1D tiling problem. The 2D tiling problem, in contrast, is undecidable [Ber66] by constructions that embed Turing machine simulations in tilings. Such embeddings of computation in tilings are what we exploit to prove interesting complexity results in 2D. At the other extreme, it seems unlikely that 3D allows for phenomena fundamentally different

extension of the theory of tiling by Wang tiles [Wan61] to include a specific mechanism for growth based on the physics of molecular self-assembly; thus it is related to physical models of crystal growth[12]. Input to the model are four elements that we refer to as a *tile system*: (1) a finite set of Wang tiles (hereafter, just tiles) with labeled sides, (2) a *strength function* on the labels specifying the strength of bonds between tiles, (3) a subset of the tiles designated as *seed tiles*[13], and (4) an integer parameter $\mathcal{T}$, the *temperature*. Defined by the model is (1) a set of *produced assemblies* — connected arrangements of tiles that can be grown from one of the seed tiles according to the rules of the model, and (2) a set of *terminal assemblies* — produced assemblies that cannot be extended according to the rules of the model. Our point of view is (1) that a tile system constitutes a *self-assembly program*, (2) that the produced assemblies represent partial computations, and (3) that the terminal assemblies are the output. Motivated by a synthetic point of view, that what matters is the number of starting materials, we consider the program-size of the tile system to be the number of tiles in the tile set.[14]

Given a tile system, the Tile Assembly Model simply defines (or predicts) the *possible* products of self-assembling an unlimited number of copies of the tiles in the tile set — it does not predict the time it takes for the products to be formed[15], nor does it predict

from 2D, since universal computation is already possible in 2D. We see a couple examples where 3D exhibits subtly different phenomena in Section 2.6

[12]Technically, because the Wang tiles may form a nonperiodic pattern, it may be classified as a model of pseudo-crystal growth.

[13]Formally, the model also allows arrangements of tiles to be designated as seed assemblies, but we do not consider using seeds larger than single tiles here.

[14]The astute reader may well ask, "Shouldn't the program-size complexity include the total amount of information in the tile system, including the strength function, seed assemblies and temperature?" Indeed the strength function contains information vital to the function of the self-assembly program, and later, when we compare the program size complexity of self-assembly programs to the program-size complexity of computer programs, we have to take this information into account. In this chapter we consider sets of seed tiles that contain only a single tile, thus the seed assemblies and the temperature add only another constant amount of information and can be ignored.

[15]For a treatment of the Tile Assembly Model augmented to deal with time complexity see [ACGH01]

the relative abundances of products that are likely to form in a real experiment. In real crystallization experiments, the range of products self-assembled from the same set of initial molecules can differ widely based on the conditions of the experiment. One example is the crystallization of a dissolved molecule from solution: if the solution is very concentrated then the result is likely to be a precipitate — a disordered aggregate of very small crystals; if the salt solution is dilute then the result is likely to be a single, large, well-ordered crystal. Similar effects can be obtained by changing the temperature and the bond strength between molecules in the crystal.

A main feature of the Tile Assembly Model is that it attempts, in a very crude way, to model such effects of experimental conditions. In kinetic models of self-assembly [Win98a], a significant portion of these effects appears to be well-modeled by combining temperature, concentration, and bond strength into a single parameter $\mathcal{T}$ — for a given absolute temperature, $\mathcal{T}$ is the amount of energy required to overcome the entropy lost when a tile attaches to a crystal divided by the energy of a unit-strength bond. In the Tile Assembly Model $\mathcal{T}$ is discrete and we refer to it as the *temperature*. Qualitatively different experimental regimes are modeled by giving $\mathcal{T}$ different integral multiples of the unit-strength bond.

As an example of how $\mathcal{T}$ may affect self-assembly, we consider the assembly of a set of seven tiles (Figure 2.2A, referred to as *counter tiles*[16]) at three different temperatures ($\mathcal{T} = 0, 1$, and 2) and give a sample assembly produced at each temperature (Figure 2.2B–D). In each case growth starts from a seed tile S (in the lower right corner of each sample

[16]The counter tiles are a self-assembly program for counting in binary, designed to count "correctly" only at $\mathcal{T} = 2$. Refer to Chapter 1, Section 1.3 for an explanation of how they count.

assembly) and proceeds by the addition of single tiles.[17] The addition of tiles is governed by the labels on their sides. Each label represents a type of binding domain, and thus each label has an associated *binding strength* which, for the tile systems considered in this chapter, must be a non-negative integer. A tile may bind a growing assembly only if the sum of the strengths of its binding interactions with the assembly are greater than or equal to $\mathcal{T}$. Thus, if $\mathcal{T} = 0$ an arbitrary tile can bind anywhere on a growing assembly — growth is completely uncontrolled (Figure 2.2B) and unrelated to whether tile labels match to form bonds (Figure 2.2E). If $\mathcal{T} = 1$ a tile must, in order to bind, match a growing assembly on

at least one side of strength 1 — growth is controlled by the interaction of a new tile with single tiles on a growing assembly (Figure 2.2C) — tiles form a single component connected by bonds (Figure 2.2F). At $\mathcal{T} = 2$ single strength-1 interactions are "too weak" to hold a new tile in place (Figure 2.2D). Analogous to the $\mathcal{T} = 1$ case, a tile may bind via a single strength-2 interaction and growth depends on interactions between the new tile and single tile on the assembly (Figure 2.2D, horizontal and vertical arrows). More interestingly, however, two or more strength-1 bonds may "cooperate" and a tile may bind into a corner (Figure 2.2D, diagonal arrows). Growth can then be controlled by the interaction of a new tile with a *pair* of tiles on the growing assembly — the bonds between tiles now form a "grid" (Figure 2.2G). In general, if we consider only tiles with strength-1 binding domains, we see that $\mathcal{T}$ measures the *cooperativity* — the number of bonds made at the same time — required for a tile to bind to an assembly. We say that binding is cooperative if the cooperativity is ≥ 1.

[17] The growth of crystals by monomer addition, as opposed to merging of crystal fragments, is a common assumption in studies of crystal growth ([Mar95], pp. 86–89); large defect-free crystals are not observed under physical conditions where growth occurs by aggregation of small fragments.

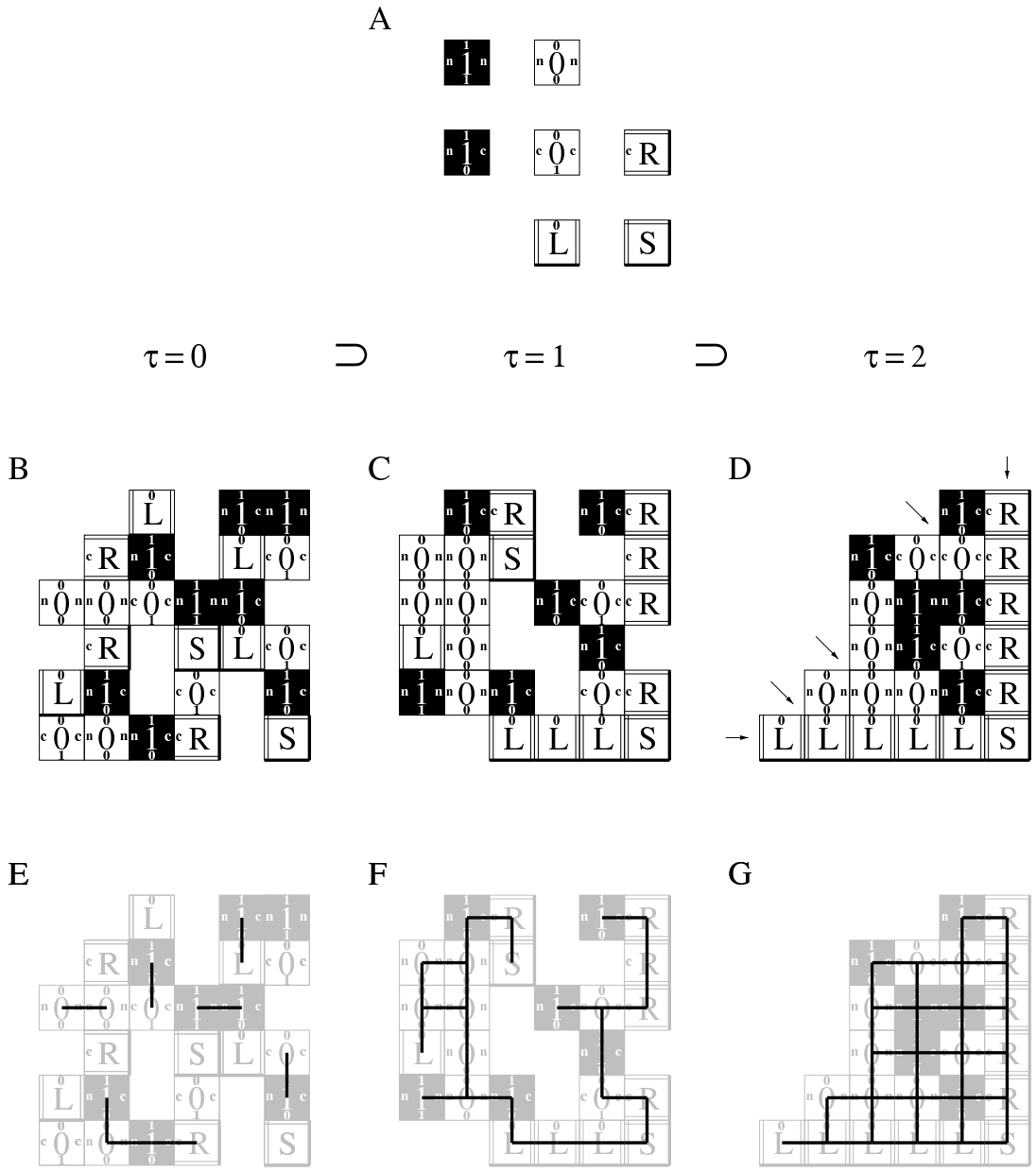


Figure 2.2: Self-assembly for different values of $\mathcal{T}$. In (A) a set T of seven counter tiles is depicted. Lines on the edges indicate bond strengths. For this figure and all figures that follow, thick lines indicate strength-0 binding domains, thin lines strength-1 binding domains, and double-lined sides strength-2 binding domains. (B)–(D) depict sample 25-tile assemblies produced by the counter tiles at $\mathcal{T} = 0, 1$ and 2 respectively. Tiles can be added: in (B) at any open side, in (C) at any side that has interaction strength 1 or greater, and in (D) positions where interactions with the new tile sum to strength 2 (indicated by arrows). The legend $\mathcal{T} = 0 \supset \mathcal{T} = 1 \supset \mathcal{T} = 2$ indicates the relationship between the sets of assemblies produced by the counter tiles as a function of temperature. For general tile sets the relationship is inclusive rather than proper. (E)–(G) show, in black, the location of bonds for the assemblies in (A)–(C).

An analogous concept of cooperativity exists in chemical systems — for them, cooperativity measures the nonlinear increase in equilibrium constant as the number of binding domains between two molecules is increased. Cooperative binding is responsible for many important phenomena in nature, from the growth of large crystals with well-defined faces, to the switching of the expression of our genes into well-defined "on" and "off' states[18]. Without cooperative binding, it is difficult to see how such phenomena could arise.

Interestingly, it *appears* that for the Tile Assembly Model, the transition between non-cooperative $\mathcal{T} = 1$ assembly to cooperative $\mathcal{T} = 2$ assembly accompanies a significant increase in computational power[19]. For $\mathcal{T} = 2$ it has been shown that one-dimensional cellular automata can simulated; hence $\mathcal{T} = 2$ self-assembly is computationally universal [Win96]. Further such cellular automata constructions for self-assembly have nice properties: they compute relatively quickly, use relatively few tile types, and produce only the desired structure. Universal computation *is possible* at $\mathcal{T} = 1$, but *so far* is limited to the simulation of Turing machines[20] in a construction that computes slowly, uses more tiles, and creates many undesired structures. In this chapter we demonstrate one *provable* difference in the computational power between $\mathcal{T} = 1$ and $\mathcal{T} = 2$: the program-size complexity of an $N \times N$ square at $\mathcal{T} = 1$ is exactly N^2 while the complexity for $\mathcal{T} = 2$ is $O(\log N)$.

[18]For a discussion of cooperative effects in gene regulation see [Pta92].

[19]It is, at this moment, unclear whether increasing $\mathcal{T}$ beyond 2 yields interesting changes in complexity. The self-assembly theory group at USC has given independent, optimal program-size and optimal time-complexity constructions for squares, both of which require only $\mathcal{T} = 2$ [ACGH01]. Until recently, the only known construction that simultaneously yielded optimal time and program-size complexity for squares required $\mathcal{T} = 3$ — now a $\mathcal{T} = 2$ construction has been found by Pablo Moisset [CdE01]. Thus, so far, we have no interesting complexity results for which $\mathcal{T} = 3$ or higher is necessary.

[20]The $\mathcal{T} = 1$ construction to which we allude was presented by Len Adleman to the USC self-assembly theory group. We discuss these issues more in Section 2.6.5.

The Tile Assembly Model seems a reasonable model, at least qualitatively, for simple natural self-assembly phenomena: $\mathcal{T} = 1$ assembly is similar to diffusion limited aggregation and $\mathcal{T} = 2$ assembly is analogous to 2D crystal growth. To embed computation in self-assembly we desire not only $\mathcal{T} = 2$ conditions but also the ability to engineer complicated sets of tiles, with many different labels (binding domains) on their sides — without such capability, the theoretical results in this paper are of limited interest. Fortunately at least two experimental systems seem suitable.

Branched DNA molecules [See98] provide a direct physical motivation for the Tile Assembly Model with $\mathcal{T} = 2$. DNA double-crossover molecules, each bearing four *sticky ends* analogous to the four sides of a tile, have been designed to self-assemble into a periodic two dimensional lattice [WLWS98b]. The binding interactions between double-crossover molecules may be redesigned by changing the base sequence of their sticky ends, thus allowing arbitrary sets of molecular *DNA tiles* to be investigated in the laboratory. Competition experiments have shown that DNA tiles matching a specially-constructed *corner site* on two sides out-compete DNA tiles matching the site on one side [Win98a] — that is, DNA tiles demonstrate $\mathcal{T} = 2$-like behaviour. Further, theoretical studies of DNA tiles using a physically-based stochastic model have shown that the Tile Assembly Model with $\mathcal{T} = 2$ is obtained in the limit of a particular binding strength/monomer concentration ratio [Win98b].

Macroscopic systems for 2D self-assembly based on lateral capillary forces [HSM96, BTCW97, BCGW99, Rot00] provide additional motivation for the Tile Assembly Model and $\mathcal{T} = 2$ assembly. In these systems millimeter-scale plastic tiles float at an interface between hydrophobic and hydrophilic liquids (*e.g.* , oil and water) and self-assemble into

lattices as the system is agitated on a shaker. Binding interactions between tiles are specified by sequences of hydrophilic and hydrophobic patches applied to the sides of tiles; when sequences match, capillary forces mediate bonds between tiles. Tile sets with up to four distinct tile types have been created by this method [Rot00]. Analogies between such systems and molecular self-assembly are not yet quantitative, but it has been observed that the frequency of shaking acts similarly to temperature and that dimers bind cooperatively to lattices. Thus cooperative $\mathcal{T} = 2$ assembly may be possible in a capillary force-based system.

One more theoretical notion, not strictly a part of the Tile Assembly Model, bears mention at this point: We say that a tile system *uniquely produces* an assembly if that assembly is the *only* terminal assembly produced by the tile system. Uniquely producing is, for us, a notion of efficiency and of correctness. Practically, a chemist considers a chemical system most efficient when it produces only a single product — if it produces too many side products in too great abundance it is impossible to purify the desired product. Theoretically, the unique production of a particular assembly is a program specification, which we must prove correct.[21] Thus our program-size results are for uniquely produced squares. We return to these issues after describing the Tile Assembly Model in greater detail.

[21] Also, I must admit, restricting our attention to tile systems that uniquely produce makes it easier (or perhaps, in some cases, possible) for us to prove statements about self-assembly.

2.3 Formal description of the Tile Assembly Model

2.3.1 Sets, tiles and configurations

We begin by reviewing the definitions of some common sets: The **natural numbers** are the set $\mathbb{N} = \{0, 1, 2, \ldots\}$, the **integers** are the set $\mathbb{Z} = \mathbb{N} \cup -\mathbb{N}$, the **positive integers** are denoted by $\mathbb{Z}^+$ and the **real numbers**, defined as usual, are denoted by the set $\mathbb{R}$. We work with the **two-dimensional grid** of integer positions, $\mathbb{Z} \times \mathbb{Z}$. The **directions** $\mathcal{D} = \{N, E, S, W\}$, used to indicate relative positions on the 2D grid, are functions from $\mathbb{Z} \times \mathbb{Z}$ to $\mathbb{Z} \times \mathbb{Z}$: $N(x, y) = (x, y + 1)$, $E(x, y) = (x + 1, y)$, $S(x, y) = (x, y - 1)$, and $W(x, y) = (x - 1, y)$. Note that $E^{-1} = W$, and $N^{-1} = S$.

A **tile** over Σ is a unit square where each side is labeled from the set Σ of **binding domains**; formally, a tile t is a 4-tuple $(\sigma_N, \sigma_E, \sigma_S, \sigma_W) \in \Sigma^4$ indicating the binding domains on the north, east, south, and west sides. For $D \in \mathcal{D}$, we write $bd_D(t)$ to refer to the binding domain of the respective side of tile t. According to this definition, rotating a tile does not usually preserve its identity; that is, $(\sigma_N, \sigma_E, \sigma_S, \sigma_W) \neq (\sigma_W, \sigma_N, \sigma_E, \sigma_S)$ unless $\sigma_N = \sigma_E = \sigma_S = \sigma_W$. The intended physical interpretation of this is (i) that tiles are not really rotationally symmetric squares — they have extra features that define an orientation and (ii) that tiles can bind to each other only if oriented in the same direction. Figure Figure 2.3 shows how extra geometric features can break symmetry and provide such an orientation. A special binding domain *null* represents a non-interaction, and the special tile $empty = (null, null, null, null)$ is used to represent the absence of any other tile.

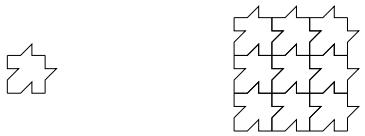


Figure 2.3: Tiles are not equivalent when rotated. One possible physical interpretation is that tiles have an orientation defined by extra geometric features. Imagine that all tiles have, in addition to their labels, the shape of the tile at left. Then tiles can only assemble if they share the same orientation, as at right.

Let T be a set of non-empty tiles. A **configuration** over T is a function $A : \mathbb{Z} \times \mathbb{Z} \rightarrow T \cup \{empty\}$ — in other words, an arrangement of tiles from T placed on an otherwise empty 2D grid. Nothing more is intended; a configuration may or may not be a possible product of self-assembly. We say that $A(x, y)$ and $A(x', y')$ are **neighbors** iff $(x', y') \in \{N(x, y), E(x, y), S(x, y), W(x, y)\}$. We say that the adjacent sides of two neighboring tiles **match** iff they are labeled with the same binding domain; otherwise they are said to **mismatch**. We say that $A(x, y)$ is **directly east** of $A(x', y')$ iff $x > x'$ and $y = y'$; similarly for the other compass directions. When used as a configuration, $\emptyset$ is called the **empty configuration**; for all (x,y), $\emptyset(x, y) = empty$. A configuration with a single non-empty tile t at position (x, y) is denoted by $\square_t^{(x,y)}$ and is known as a **tile configuration.**

Addition of configurations A and B is defined by $C = A + B$ where

$$C(x, y) = \begin{cases} A(x, y) & \text{if } B(x, y) = empty \\ B(x, y) & \text{if } A(x, y) = empty \\ \infty & \text{otherwise.} \end{cases}$$

Note that C is not necessarily a configuration, because C might contain ∞ values. If $C = A + B$, we also say that $A \subseteq C$.

Union of configurations A and B is defined by $C = A \cup B$ where

$$C(x,y) = \begin{cases} A(x,y) & \text{if } A(x,y) = B(x,y) \text{ or } B(x,y) = \textit{empty} \\ B(x,y) & \text{if } A(x,y) = B(x,y) \text{ or } A(x,y) = \textit{empty} \\ \infty & \text{otherwise.} \end{cases}$$

Note that C is not a configuration iff there is a site (x,y) s.t. $A(x,y)$ and $B(x,y)$ are distinct non-empty tiles.

Intersection of configurations A and B is defined by $C = A \cap B$ where

$$C(x,y) = \begin{cases} A(x,y) & \text{if } A(x,y) = B(x,y) \\ \textit{empty} & \text{if } A(x,y) = \textit{empty} \text{ or } B(x,y) = \textit{empty} \\ \infty & \text{otherwise.} \end{cases}$$

Note that C is not a configuration iff there is a site (x,y) s.t $A(x,y)$ and $B(x,y)$ are distinct non-empty tiles.

2.3.2 Strength functions, temperature and assemblies

Given a configuration, a **strength function** over binding domains is used to determine the strength of the interaction between neighboring tiles; that is, how strongly two tiles stick to each other. In general, a strength function is a function $g : \Sigma \times \Sigma \to \mathbb{R}$, where $\textit{null} \in \Sigma$, is a strength function iff $\forall \sigma, \sigma' \in \Sigma$, $g(\sigma, \sigma') = g(\sigma', \sigma)$ and $g(\textit{null}, \sigma) = 0$. A **diagonal**

strength function is a strength function that is non-zero only for $g(\sigma, \sigma')$ such that $\sigma \neq \sigma'$. A **non-negative** strength function is a strength function such that $g(\sigma, \sigma') \geq 0$ for all σ, σ'. Here we will mostly consider non-negative diagonal strength functions; we discuss some implications of non-diagonal strength functions and strength functions with negative values in Section 2.6.

Within a configuration A, we say that the tile at (x, y) and its neighbor in direction D, $D(x, y)$, interact with a strength

$$\Gamma_D^A(x, y) = g(bd_D(A(x, y)), bd_{D-1}(A(D(x, y)))).$$

If $\Gamma_D^A(x, y) > 0$, then the tiles **bind** or **make a bond.** We say the bond is **weak** iff $\Gamma_D^A(x, y) \leq 1$; the bond is a **strong** iff $\Gamma_D^A(x, y) > 1$. If no bond is made the tiles have a **null interaction.**

The **free energy** of a configuration C is the sum of all interaction strengths between tiles (in contrast to standard usage in chemistry, favorable interactions are given by positive numbers):

$$G(C) = \frac{1}{2} \sum_{x,y \in \mathbb{Z}} \sum_{D \in \mathcal{D}} \Gamma_D^C(x, y).$$

The **temperature** $\mathcal{T}$ is analogous to the average thermal energy in a physical system; in a physical system objects bound by interactions with a strength greater than the average thermal energy are stable, those bound by interactions with a strength is less than the average thermal energy fall apart. Thus we say that a configuration C is a **$\mathcal{T}$-stable assembly** iff for all non-empty configurations A and B such that $C = A + B$, $G(C) \geq G(A) + G(B) + \mathcal{T}$. In other words, a $\mathcal{T}$-stable assembly cannot fall apart into two pieces

without decreasing the total G by $\mathcal{T}$ or more. Note that for $\mathcal{T} > 0$, a $\mathcal{T}$-stable assembly must contain a single connected component. When $\mathcal{T}$ is understood, we simply say that C is an assembly. The **size** of an assembly C, denoted by $|C|$, is the number of non-empty positions (x, y) in C. An assembly is a **full assembly** iff it has no null interactions between neighboring non-empty tiles.

2.3.3 Tile systems and the posets of assemblies they produce.

A **tile system T** is specified by the quadruple $\langle T, \mathcal{S}, g, \mathcal{T} \rangle$, where T is a finite set of non-empty tiles, $\mathcal{S}$ is a set of $\mathcal{T}$-stable **seed assemblies** over T, g is a strength function, and $\mathcal{T} \in \mathbb{R}$ is the temperature. We say that **T** is a **discrete** tile system iff $g \to \mathbb{N}$, $\mathcal{T} \in \mathbb{N}$, and $\mathcal{S}$ contains only tile configurations. We say that a discrete tile system is **singly seeded** iff $\mathcal{S}$ contains only one element, the tile configuration $S = \square_s^{(0,0)}$ for some **seed tile** s. We say that a discrete tile system is **unseeded** iff $\mathcal{S} = \{\square_t^{(0,0)}$ for all $t \in T\}$; then all $t \in T$ are seed tiles. We also say that an assembly A is **uniquely seeded** iff A contains only a single occurrence of s.

Self-assembly is defined by a relation between configurations: $A \to_{\mathbf{T}} B$ iff there exists a tile $t \in T$ and a site (x, y) such that $B = A + \square_t^{(x,y)}$ and B is $\mathcal{T}$-stable. Since $G(\square_t^{(x,y)}) = 0$, $G(B) \geq G(A) + \mathcal{T}$; *i.e.* , a tile **can be added** to an assembly iff the summed strength of its interactions with its neighbors exceeds a threshold set by the temperature. In particular, for discrete tile systems with $\mathcal{T} = 1$, a tile can be added if it makes *any* bond to a neighbor, whereas for discrete tile systems with $\mathcal{T} = 2$, to be added the tile must either make two weak bonds or a single strong bond. The relation $\to_{\mathbf{T}}^*$ is the reflexive transitive closure of $\to_{\mathbf{T}}$ and we say A **assembles to** B or equivalently B **assembles**

from A iff $A \rightarrow_{\mathbf{T}}^{*} B$. Given $A \rightarrow_{\mathbf{T}}^{*} B$, an **assembly sequence** is a sequence of single tile additions from A to B; we also call A a **subassembly** of B.

Recall that a **partially ordered set** or **poset** $\langle \mathcal{P}, \leq \rangle$ is a set $\mathcal{P}$ and a reflexive, transitive, antisymmetric relation $\leq$. Thus, the tile system and relation $\rightarrow_{\mathbf{T}}^{*}$ define a poset comprised of a set, the **produced** assemblies $Prod(\mathbf{T})$:

$$Prod(\mathbf{T}) = \{A \text{ s.t. } \exists S \in \mathcal{S} \text{ s.t. } S \rightarrow_{\mathbf{T}}^{*} A\}$$

and the relation

$$A \leq B \text{ iff } A \rightarrow_{\mathbf{T}}^{*} B.$$

We note that for $\mathbf{T}_1 = \langle T, \mathcal{S}, g, \mathcal{T}_1 \rangle$ and $\mathbf{T}_2 = \langle T, \mathcal{S}, g, \mathcal{T}_2 \rangle$ and $\mathcal{T}_1 < \mathcal{T}_2$ it is easy to see that $Prod(\mathbf{T}_2) \subseteq Prod(\mathbf{T}_1)$.

We say that an assembly A is **maximal** iff $\nexists B$ s.t. $A < B$. ($A < B$ is defined as usual, *i.e.* $A < B$ iff $A \leq B$ and $A \neq B$.) Thus another set, the **terminal** assemblies $Term(\mathbf{T})$, is defined as the maximal elements of $Prod(\mathbf{T})$:

$$Term(\mathbf{T}) = \{A \in Prod(\mathbf{T}) and \nexists B \text{ s.t. } A < B\}.$$

We also say that an assembly A is **maximal for some predicate** P iff $\nexists B$ s.t. $A < B$ and B satisfies P. **Minimal** assemblies are defined similarly.

The produced assemblies include intermediate products of the self-assembly process, whereas the terminal assemblies are just the end products, and may be considered the output. A tile system $\mathbf{T}$ is **haltable**, in the sense that every sequence of self-assembly

can eventually terminate, iff $\forall A \in Prod(\mathbf{T}) : \exists B \in Term(\mathbf{T})$ s.t. $A \rightarrow_{\mathbf{T}}^{*} B$. A tile system $\mathbf{T}$ is **halting**, in the sense that every sequence of self-assembly *does* eventually terminate, iff $\mathbf{T}$ is haltable and $Term(\mathbf{T})$ is finite. A tile system $\mathbf{T}$ **uniquely produces** A iff $Term(\mathbf{T}) = \{A\}$ and $\mathbf{T}$ is halting. A tile system $\mathbf{T}$ **exactly produces** a finite set of assemblies $\mathcal{A}$ iff $Term(\mathbf{T}) = \mathcal{A}$ and $\mathbf{T}$ is halting. $\mathcal{U}P$ is the set of all tile systems that uniquely produce *some assembly* A. $\mathcal{U}P^{A}$ is the set of all pairs $\langle \mathbf{T}, A \rangle$ such that $\mathbf{T}$ uniquely produces A.

Understanding $\langle Prod(\mathbf{T}), \leq \rangle$ as a poset is sometimes instructive — recall the following for a poset $\langle \mathcal{P}, \leq \rangle$: For m, a, and $b \in \mathcal{P}$, m is called the **meet** of a and b iff $m \leq a$ and $m \leq b$ and $\forall c \in \mathcal{P} : [c \leq a \text{ and } c \leq b] \implies c \leq m$. Thus, the meet m of a and b is the unique maximal assembly such that $[m \leq a \text{ and } m \leq b]$ — if there is more than one such maximal assembly, no meet exists. For j, a, and $b \in \mathcal{P}$, j is called the **join** of a and b iff $a \leq j$ and $b \leq j$ and $\forall c \in S : [a \leq c \text{ and } b \leq c] \implies j \leq c$. Thus, the join j of a and b is the unique minimal assembly such that $[a \leq j \text{ and } b \leq j]$ — if there is more than one such minimal assembly, no join exists.

The poset $\langle \mathcal{P}, \leq \rangle$ is called a **lattice** iff all pairs $a, b \in \mathcal{P}$ have both a meet and a join. An interesting observation is that if a tile system uniquely produces C, then $\langle Prod(\mathbf{T}), \leq \rangle$ is a lattice: the join of A and B is $A \cup B$, and the meet of A and B is the maximal C' such that $[C' \subseteq (A \cap B) \text{ and } C' \in Prod(\mathbf{T})]$. In general, if $\langle Prod(\mathbf{T}), \leq \rangle$ is a lattice, we say that $\mathbf{T}$ produces a **unique pattern** — $\mathbf{T}$ need not be halting nor even haltable. By our definition of the assembles relation, it is easy to see that for all singly seeded tile systems with non-negative strength functions, a meet exists for all A, B in $Prod(\mathbf{T})$ (Lemma 3).

Thus to show that such a tile system produces a unique pattern it is sufficient to show that a join exists for all A, B in $Prod(\mathbf{T})$.

2.3.4 Translation invariant assemblies

As defined, an assembly is a coordinate system specific object, and, as such, it is useful for proving theorems about self-assembly. However, our real interest is in assemblies of tiles independent of their coordinate system. We will see that as long as the tile systems that we are interested in are singly seeded, and all the assemblies produced are uniquely seeded, all of our results for assemblies transfer gracefully to a coordinate system independent view. However, if we wish to consider unseeded tile systems or multiply seeded assemblies this is not true — for example, a tile system with multiple seed tiles may produce only assemblies that are equivalent under translation but, as defined, the tile system *does not* uniquely produce. (See Figure 2.7a.)

To treat more general tile systems we follow the spirit of [ACGH01] and define a **translation-invariant assembly** (or *ti*-**assembly**) to be an equivalence class of assemblies under translation. The function " $\widetilde{\ }$ " is a function that maps assemblies to *ti*-assemblies. In particular " $\widetilde{\ }$ ", maps an assembly A to the class of assemblies equivalent to A under translation; we write $\widetilde{\ }(A)$ as $\widetilde{A}$. *ti*-assemblies inherit most definitions from assemblies easily: In general, for a coordinate system independent property X, we say that *ti*-assembly $\widetilde{A}$ has property X iff there exists an assembly $A' \in \widetilde{A}$ with property X. Relations are also defined analogously: $\widetilde{A} \rightarrow_{\mathbf{T}}^{*} \widetilde{B}$ iff $\exists A' \in \widetilde{A}$ and $B' \in \widetilde{B}$ such that $A' \rightarrow_{\mathbf{T}}^{*} B'$. Similarly, $\widetilde{Prod(\mathbf{T})}$ is the set $\{\widetilde{A} : \exists A' \in \widetilde{A} \text{ s.t. } A' \in Prod(\mathbf{T})\}$ and $\widetilde{Term(\mathbf{T})}$ is the set $\{\widetilde{A} : \exists A' \in \widetilde{A} \text{ s.t. } A' \in Term(\mathbf{T})\}$

A tile system $\mathbf{T}$ **uniquely produces** $\widetilde{A}$ iff $\widetilde{Term(\mathbf{T})} = \{\widetilde{A}\}$ and $\mathbf{T}$ is halting. The set $\widetilde{\mathcal{UP}}$ is, as expected, the set of tile systems that uniquely produce *some ti*-assembly $\widetilde{A}$. The set $\widetilde{\mathcal{UP}}^A$ is the set of pairs $\langle \mathbf{T}, A \rangle$ such that $\mathbf{T}$ uniquely produces $\widetilde{A}$. Obviously $\widetilde{\mathcal{UP}} \subseteq \mathcal{UP}$ and $\widetilde{\mathcal{UP}}^A \subseteq \mathcal{UP}^A$. Thus if we prove "$\mathbf{T}$ uniquely produces A" we have shown "$\mathbf{T}$ uniquely produces $\widetilde{A}$".

Occasionally we use the $\subseteq$ relation on *ti*-assemblies. Thus, for the *ti*-configuration $\widetilde{C}$ we write $\widetilde{C} = A + B$ to mean the class of configurations equivalent to $C = A + B$. We define the set $\mathbb{C} = \widetilde{A} + \widetilde{B}$ to be the set of equivalence classes $\widetilde{C} = A' + B'$ for all $A' \in \widetilde{A}$ and $B' \in \widetilde{B}$ such that $A + B$ is a configuration. We say that $\widetilde{A} \subseteq \widetilde{C}$ iff there exists an $A' \in \widetilde{A}$ and a configuration B such that $\widetilde{C} = A' + B$. Or equivalently $\widetilde{A} \subseteq \widetilde{C}$ iff there exist a *ti*-configuration $\widetilde{B}$ such that $\widetilde{C} \in \mathbb{C} = \widetilde{A} + \widetilde{B}$.

To prove statements about *ti*-assemblies, we prove statements about assemblies. In particular, we prove statements about assemblies with special coordinate systems: an assembly A, with respect to a tile system $\mathbf{T}$, is a **seed-aligned** assembly iff $A(0,0) = S(0,0)$ for some $S \in \mathcal{S}$. By definition, *all* produced assemblies are seed-aligned assemblies.

An important question is: "When do results that we have proven for produced assemblies of a tile system hold for the produced *ti*-assemblies of the tile system?" It is clear that results will transfer if the two posets are the same up to a re-labeling of their elements. Thus we define a notion of poset isomorphism: we say that two posets $\langle \mathcal{P}_1, \leq_1 \rangle$, $\langle \mathcal{P}_2, \leq_2 \rangle$ are **isomorphic** iff there exists a function $f : \mathcal{P}_1 \to \mathcal{P}_2$ such that $\forall A, B \in \mathcal{P}_1$:

1. $A = B$ iff $f(A) = f(B)$
2. $A \leq_1 B$ iff $f(A) \leq_2 f(B)$

We now prove a *sufficient* condition for isomorphism between the produced assemblies and produced *ti*-assemblies:

Theorem 1 $\forall \mathbf{T}$, *where* $\mathbf{T}$ *is a singly seeded discrete tile system and,* $\forall A \in Prod(\mathbf{T}), A$ *is a uniquely seeded assembly:* $\langle Prod(\mathbf{T}), \leq \rangle$ *and* $\langle \widetilde{Prod(\mathbf{T})}, \leq \rangle$ *are isomorphic.*

PROOF. We show that a function from $Prod(\mathbf{T}) \rightarrow \widetilde{Prod(\mathbf{T})}$ satisfying the definition of poset isomorphism is just the function "$\widetilde{\ }$". Observe first that, by definition of $\widetilde{Prod(\mathbf{T})}$, $\forall A, B \in Prod(\mathbf{T}) : \widetilde{A}, \widetilde{B} \in \widetilde{Prod(\mathbf{T})}$. Thus "$\widetilde{\ }$" : $Prod(\mathbf{T}) \rightarrow \widetilde{Prod(\mathbf{T})}$

Now we show $\forall A, B \in Prod(\mathbf{T}) : A = B$ iff $\widetilde{A} = \widetilde{B}$. **(only if)** Trivial by definition of "$\widetilde{\ }$". **(if)** Assume not. Then $\widetilde{A} = \widetilde{B}$ and $A \neq B$. If this is the case then A and B differ only by translation. Either:

1. $A(0,0) = B(0,0)$, that is there are two occurrences of a single seed tile in A and $A \in Prod(\mathbf{T})$ is not uniquely seeded. Contradiction. OR

2. $A(0,0) \neq B(0,0)$, that is there exist two different tiles that are seed tiles (or A and B could not be produced). $\mathbf{T}$ not singly seeded. Contradiction.

Now we show $\forall A, B \in Prod(\mathbf{T}) : A \leq B$ iff $\widetilde{A} \leq \widetilde{B}$. **(only if)** Trivial by the definitions of "$\widetilde{\ }$" and $\leq$ for *ti*-assemblies. **(if)** Assume not. Then (i) $\widetilde{A} \leq \widetilde{B}$ and (ii) $A \not\leq B$. By (i) there exists an $A' \in \widetilde{A}$ and a $B' \in \widetilde{B}$, $A', B' \in Prod(\mathbf{T})$ such that $A' \leq B'$. Since A and A' (or B and B') are in the same equivalence classes, they can only differ by translation. By (ii) it cannot be the case that $[(A = A')$ and $(B = B')]$, that is, $[(A \neq A')$ or $(B \neq B')]$. Assume that $A \neq A'$. Then by arguments (1) and (2) above we reach contradictions if either $A(0,0) = A'(0,0)$ or $A(0,0) \neq A'(0,0)$. Similar for $B \neq B'$. ■

Theorem 1 seems obvious — and it is. Assume for the moment that the Tile Assembly Model is a perfect model for self-assembly in the real world. Theorem 1 simply reassures us that if we are pondering a tile system (for which we have proven the preconditions for the theorem), and are enumerating produced assemblies we do not need to bother checking whether any two produced assemblies are equivalent under translation — if two produced assemblies are different they actually correspond to two different chemical species in the real world.

Note that the converse of Theorem 1 is not true; there exist both multiply seeded tile systems, and tile systems producing non-uniquely seeded assemblies for $Prod(\mathbf{T})$ and $\widetilde{Prod(\mathbf{T})}$ *are* isomorphic. Figure 2.7c and Figure 2.8c are examples of the latter. In addition, we will sometimes be interested in tile systems for which the mapping between $Prod(\mathbf{T})$ and $\widetilde{Prod(\mathbf{T})}$ is not isomorphic. For such tile systems we will note explicitly those places where results on assemblies do not directly transfer to *ti*-assemblies. In general, such tile systems force us to consider more than a single coordinate system for each assembly of interest — with the effect of complicating our proofs.

2.3.5 Lemmas for non-negative strength functions

For many of our proofs we make use of the special nature of the strength functions that we study — in particular that the bond strengths are always non-negative. We refer to this property as the **non-negativity of bond strengths** or just **non-negativity.** The physical interpretation of non-negativity is that interactions between tiles are always attractive[22]. Non-negativity insures that if a tile t can bind to a site (x, y) on an assembly

[22]See Section 2.6.4 for a discussion of the physical and mathematical implications of repulsive interactions between tiles.

C, then no matter what tiles are added to C at other sites, t can always bind at site (x, y), *i.e.* :

Lemma 1 *$\forall \langle \mathbf{T}, C, A \rangle$, $\mathbf{T} = \langle T, \mathcal{S}, g, \mathcal{T} \rangle$ is a tile system s.t. g is non-negative and C, A are assemblies s.t. $C \subseteq A$: [$C \rightarrow_{\mathbf{T}} C + \square_t^{(x,y)}$ and $A(x, y) = empty$] implies $A \rightarrow_{\mathbf{T}} A + \square_t^{(x,y)}$.*

PROOF. Assume not. Then $C \rightarrow_{\mathbf{T}} C + \square_t^{(x,y)}$ and there exists an $A \supseteq C$ s.t. $A(x, y)$ and $A \not\rightarrow_{\mathbf{T}} A + \square_t^{(x,y)}$. Let the sum of the interactions g between C and $\square_t^{(x,y)}$ be I_{C-t}, between A and $\square_t^{(x,y)}$ be I_{A-t}, and between any tiles non-empty in A but empty in C with $\square_t^{(x,y)}$ be $I_{\overline{C}-t}$. By assumption $I_{C-t} \geq \mathcal{T}$ and $I_{A-t} < \mathcal{T}$. Since A has all the tiles of C, $I_{A-t} = I_{C-t} + I_{\overline{C}-t}$, and we must have $I_{\overline{C}-t} < 0$. At least one of the interactions in g summed to get $I_{\overline{C}-t}$ must be negative contradicting our assumption of non-negativity . ■

Non-negativity can be used to prove other useful lemmas. For example, while [$C \leq A$ *implies* $C \subseteq A$] the converse [$C \subseteq A$ *implies* $C \leq A$] is not true. However, we *can* show:

Lemma 2 *$\forall \langle \mathbf{T}, C, A \rangle$, $\mathbf{T} = \langle T, \mathcal{S}, g, \mathcal{T} \rangle$ is a tile system s.t. g is non-negative and C, A are assemblies: [$A, C \in Prod(\mathbf{T})$ and $C \subseteq A$] implies $C \leq A$.*

PROOF. $C \in Prod(\mathbf{T})$ so we know that there exists an assembly sequence for C: $S \rightarrow_{\mathbf{T}} C_1 \rightarrow_{\mathbf{T}} \ldots \rightarrow_{\mathbf{T}} C_{|C|} = C$. Let the tile configurations added at each step to assemble C according to this sequence be $c_1, \ldots, c_{|C|}$. $A \in Prod(\mathbf{T})$ so we know that there exists an assembly sequence for A: $S \rightarrow_{\mathbf{T}} A_1 \rightarrow_{\mathbf{T}} \ldots \rightarrow_{\mathbf{T}} A_{|A|} = A$. The tile configurations added in this assembly sequence include the configurations c, plus tile configurations $\bar{c}$ for all tiles from A that occur at *empty* positions in C. In the assembly sequence for A the

configurations c and $\bar{c}$ are added in some order, but we label the $\bar{c}$ in ascending orders, for example: $\bar{c_1}, c_5, \bar{c_2}, c_{|C|}, \ldots, c_1, \bar{c}_{|A|-|C|}$. Let the assembly just before the addition of $\bar{c}_1$ be $\bar{A}_0$ and let each assembly just after the addition of $\bar{c}_i$ be $\bar{A}_i$.

Now, we can construct a new assembly sequence from C to A that makes the additions $\bar{c}_1, \ldots, \bar{c}_{|A|-|C|}$. Starting with C, let the sequence of assemblies that make these additions be $C = \bar{A}'_0, \ldots, \bar{A}'_{|A|-|C|}$. We show these steps can be made by induction. The base case is a simplification of the inductive step. Consider an addition $\bar{c}_i$, that is, the step $\bar{A}'_i = \bar{A}'_{i-1} + \bar{c}_i$. Any tiles from C that are required to add the tile configurations $\bar{c}_i$ are already in place since all tiles in C are present; likewise, because we add the $\bar{c}$ in order as before, any $\bar{c}$ required for $\bar{c}_i$ to be added are already in place; finally, by non-negativity, any tiles occurring in $\bar{A}'_{i-1}$ that do not occur in $\bar{A}_{i-1}$ do not interfere with the addition of $\bar{c}_i$. ■

Non-negativity can also be used to show a special feature of the posets that arise from self-assembly. Specifically:

Lemma 3 *$\forall \langle \mathbf{T}, A, B \rangle$, $\mathbf{T} = \langle T, \mathcal{S}, g, \mathcal{T} \rangle$ is a tile system s.t. g is non-negative, $A, B \in Prod(\mathbf{T})$ are assemblies, A is finite, and $A(0,0) = B(0,0)$: There exists a meet $C \in Prod(\mathbf{T})$ for A and B.*

PROOF. A meet C is the unique maximal assembly such that $P = [C \in Prod(\mathbf{T})$ and $C \leq A$ and $C \leq B]$. Let S be the element of $\mathcal{S}$ such that $S(0,0) = A(0,0)$. We know that a maximal assembly satisfying P exists since (i) since A serves as an upper bound for an assembly satisfying P and (ii) $S \leq A$ and $S \leq B$ — if no greater assembly assembly satisfies P then S is maximal for P.

Now we prove C is unique by contradiction. Assume $D \neq C$ exists and is maximal for P. Consider E, a maximal assembly such that $Q = [E \in Prod(\mathbf{T})$ and $E \leq C$ and $E \leq D]$. E exists because (i) A serves as an upper bound for E and (ii) S satisfies Q. Clearly $E < C$, otherwise $C \leq D$ violating the maximality of C for P. Thus, a tile from C can be added to E: there exists a nonempty tile u and position (i, j) such that $E + \square_u^{(i,j)} \leq C$. Further, $E + \square_u^{(i,j)} \not\leq D$; otherwise would violate the maximality of E for Q. Thus C and D disagree at (i, j) — wherever C and D disagree one is empty and the other is not otherwise they could not both assemble to A. Thus D must be empty at (i, j). Since $E \subseteq D$, the non-negativity of bond strengths (by Lemma 1 and $D \in Prod(\mathbf{T})$) means that $D' = D + \square_u^{(i,j)}$ must be in $Prod(\mathbf{T})$. Further since $D' \subseteq A$, $D' \subseteq B$, and $A, B \in Prod(\mathbf{T})$, Lemma 2 implies $D' \leq A$ and $D' \leq B$; thus D' violates the maximality of D for P ■

In particular, Lemma 3 applies to singly seeded discrete tile systems because, for such tile systems, all elements of $Prod(\mathbf{T})$ share the same seed tile.

2.4 Efficiency and Correctness for Self-assembly Programs

To help motivate notions of efficiency and correctness for self-assembly programs, we revisit the counter tiles Figure 2.4. At $\mathcal{T} = 2$, these tiles count in binary; the n^{th} row above the origin represents the binary integer n. The counter tiles are analogous to an infinite loop — no terminal assemblies are produced.

The counter tiles have two nice properties. The first is that they constitute a very small program (just seven tiles) for enumerating all binary numbers. Thus the counter tiles satisfy our intuitive notion of *tile-efficiency*; that is, a self-assembly program should

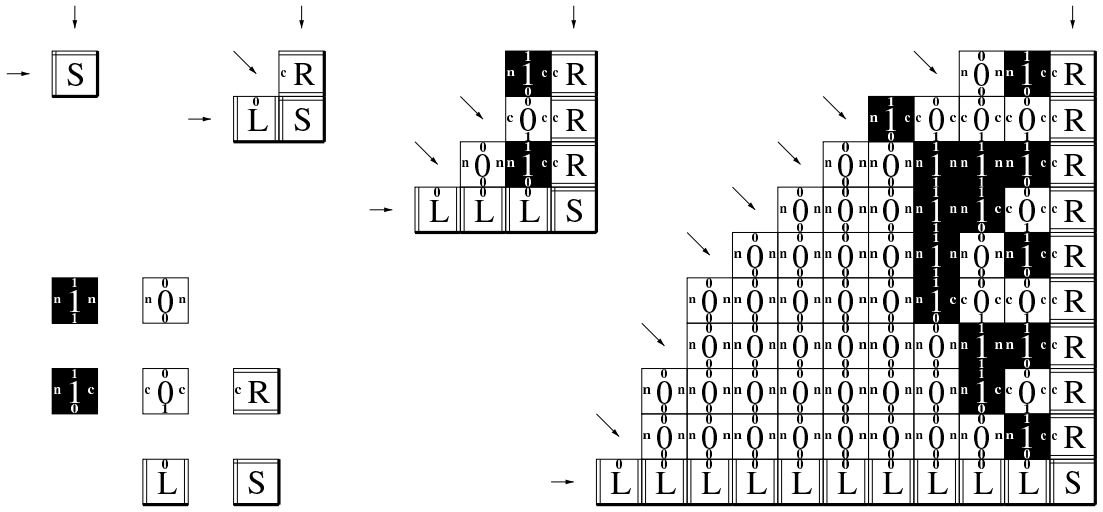


Figure 2.4: Simulating a binary counter with self-assembly. Bottom left, a set T of seven tiles is depicted. From top left to top right, successively larger assemblies produced by $\mathbf{T} = \langle T, \{s\}, g, 2 \rangle$ are shown. Arrows indicate positions at which they can grow. Numbers at far right are the decimal interpretation of rows in the assembly.

use a small number of types of tiles. The second nice property of the counter tiles is that the pattern created by the tiles is unique: all assemblies that are produced are consistent with a single, infinite pattern. Thus the counter tiles also satisfy our intuitive notion of *assembly-efficiency*; that is, a self-assembly program should not waste tiles creating assemblies other than the assemblies that are desired. Further, proof that the counter tiles produce a single unique pattern would be a proof of *correctness* — that the tiles behave as expected. In the subsections that follow we define tile-efficiency and assembly-efficiency, as well as give algorithms for proving that self-assembly programs are correct.

2.4.1 Tile-efficiency and uniquely addressed assemblies

Tile-efficiency[23], properly conceived, is a relative thing — we say that a tile system $\mathbf{T}_1$ that produces assembly A using $|T_1|$ types of tiles is **more tile-efficient** than another program $\mathbf{T}_2$ that produces A using $|T_2|$ types of tiles iff $|T_1| < |T_2|$. More generally, for an infinite sequence of assemblies $\mathcal{A} = \{A_1, A_2, \ldots\}$ of increasing size, we say that an infinite sequence of tile systems[24] $\mathbb{T}_1 = \{\mathbf{T}_1, \mathbf{T}_2, \ldots\}$ producing the elements of $\mathcal{A}$ is **more tile-efficient** than another infinite sequence of tile systems $\mathbb{T}_2$ producing $\mathcal{A}$, iff $\mathbb{T}_1$ uses an asymptotically smaller number of tiles than $\mathbb{T}_2$ as a function of the size of assemblies in $\mathcal{A}$.

In designing self-assembly programs, we would like to have some guidance as to whether our constructions are "tile-efficient enough" to be realized physically. Thus, even with an understanding that tile-efficiency is properly a relative concept, it is still of interest to pick a program size (as a function of assembly size) below which a self-assembly program is to be considered tile-efficient and thus "practical", and above which, the program is considered tile-inefficient and thus "impractical". Making such a choice is difficult because, experimentally, we are still deciding what self-assembly programs are practical. So far, it is clear that one class of self-assembly programs *is* asymptotically impractical — those that use a tile set that is equal or greater in size to the assembly of interest. Since we design self-assembly programs with a particular target assembly in mind, we express this idea in terms of assemblies; that is we say that an assembly A is **uniquely addressed** iff the

[23] The same as, but more euphonious and evocative than, program-size efficiency.

[24] All of the constructions that we give for squares in Section 2.5 are, in fact, descriptions of infinite sequences of tile systems for infinite sequences of assemblies (squares of increasing size). For some discussions of complexity, as above, it makes sense to identify the idea of a "self-assembly program" with such an infinite sequence of tile systems rather than a single tile system.

number of distinct types of non-empty tiles in A equals $|A|$ — any tile system producing A must have $|T| \geq |A|$. We also sometimes use uniquely addressed informally to describe some part of a larger assembly if, for the part in question, all of the tiles are distinct and they are used nowhere else in the larger assembly. An example of a uniquely addressed assembly is shown in Figure 2.12a.

In the lab, it is only practical to create relatively small uniquely addressed assemblies. This is not only because is it costly to buy or synthesize $|A|$ different starting materials: In a chemical system the rate at which a tile binds is proportional to its concentration and, for uniquely addressed assemblies, each tile species can represent at most $1/|A|$ of the total concentration of tiles. In practice, the total concentration of tiles has a fixed upper limit. Thus, for uniquely addressed assemblies of increasing size, the rate at which tiles bind drops quickly; large uniquely addressed assemblies may take an impractically long time to grow.

Unfortunately, there are instances in which one cannot avoid using uniquely addressed assemblies: to synthesize a one-dimensional assemblies exactly N tiles long *requires* N distinct tiles. Using a process closely related to self-assembly, "uniquely addressed" one-dimensional DNA structures with the equivalent of about one hundred distinct tiles [SCH+95] have been created but we would be surprised if such methods could be scaled up to create uniquely addressed structures with 10000 distinct tiles.

Given that uniquely addressed assemblies use an impractical number tiles, what are we to consider a practical number of tiles? We believe that it will be practical to create relatively large square assemblies for which one edges is uniquely addressed but the rest

of the square is composed of a constant number of distinct tiles (see Figure 2.14b) — using such a scheme we can imagine synthesizing 100 different chemical tiles to create an assembly of 10000 tiles in size. Thus we propose that tile-efficient programs be allowed to specify at most a one-dimensional fraction of a two dimensional assembly (in our example, the edge of the square). For a single tile system we say $\mathbf{T}$ is **tile-efficient** for assembly A iff $A \in Prod(\mathbf{T})$ and $|T| \leq \sqrt{|A|}$. More generally, we say that a infinite sequence of tile systems $\mathbb{T} = \{\mathbf{T}_1, \mathbf{T}_2, \ldots\}$ is tile-efficient for an infinite sequence of assemblies $\mathcal{A} = \{A_1, A_2, \ldots\}$ of increasing size iff $A_n \in Prod(\mathbf{T}_n)$ and $|T_n| = O(\sqrt{|A_n|})$. If A or $\mathcal{A}$ is understood we just say that $\mathbf{T}$ or $\mathbb{T}$ is tile-efficient. It isn't clear yet whether these are useful definitions of tile-efficient; at least they excludes tile systems that produce uniquely addressed assemblies.

2.4.2 Assembly-efficiency and uniquely produced assemblies

Let us return again to the counter tiles, and consider and consider the set of *ti*-assemblies produced by the counter tiles at $\mathcal{T} = 2$, Bin^2. Although we have not given a proof (essentially a simplification of Lemma 12 on p. 96), we have asserted that all assemblies in Bin^2 are consistent with the binary counter pattern — they are for us, desirable *ti*-assemblies. And, because there are just seven counter tiles, all of the elements Bin^2 with more than 49 tiles are self-assembled tile-efficiently. Now consider the sets Bin^1 and Bin^0 of *ti*-assemblies produced by the counter tiles at $\mathcal{T} = 1$ and $\mathcal{T} = 0$. $Bin^2 \subset Bin^1 \subset Bin^0$; at lower temperatures all of the assemblies we care about, Bin^2, are still produced — if we believe that lower temperatures are easier to achieve experimentally (See Section 2.6.5, why not assemble the counter tiles at $\mathcal{T} = 1$ or $\mathcal{T} = 0$? Consider the number of N-tile

number of tiles per *ti*-assembly	1	2	3	4
number of *ti*-assemblies: $\mathcal{T} = 2$	1	2	3	5
$\mathcal{T} = 1$	1	2	7	39
$\mathcal{T} = 0$	1	26	762	20995

Table 2.1: Number of produced *ti*-assemblies with a given number of counter tiles as a function of temperature $\mathcal{T}$.

ti-assemblies produced as a function of temperature, Table 2.1. The number of N-tile *ti*-assemblies in Bin^2 increases as $O(e^{\alpha\sqrt{N}}/N)$, while the number of assemblies in Bin^1 and Bin^0 increase at least exponentially, as $\Omega(2^N)$ and $\Omega(3.7^N)$, respectively.[25] If we imagine a physical system that implements the assembly of counter tiles at $\mathcal{T} = 1$ or $\mathcal{T} = 0$ we see that our desired assemblies would get buried in an avalanche of undesired ones.

The lesson is that even though an abstract tile system $\mathbf{T}$ may produce an assembly A that we desire tile-efficiently, we may not be able to recover A from an equivalent physical system in any effective way — we have to consider the other assemblies that are also produced by the tile system. One might be tempted to try to "purify" $Prod(\mathbf{T})$ and extract just those elements that were desired. In general, however, purification is physically difficult[26] and economically infeasible — any unwanted assemblies formed and removed

[25]The number of *ti*-assemblies at $\mathcal{T} = 2$ is just the additive partition of N, $p(N) \sim \frac{e^{\pi\sqrt{2/3}\sqrt{N}}}{4\sqrt{3}N}$ as shown by Hardy and Ramanujan [IK80]. The *ti*-assemblies at $\mathcal{T} = 1$ were counted by hand but there are at least 2^N of them because we can make N-tile "worms" for which we can always choose to add a tile to the west or the north. No closed form for the number of $\mathcal{T} = 0$ *ti*-assemblies is known, but we can give a compact expression. Let G be the number of connected geometrical arrangements of N tiles, s be the number of copies of the seed tile in a *ti*-assembly and X be the number of non-seed tiles. Then the number of $\mathcal{T} = 0$ *ti*-assemblies was calculated as $G\sum_{s=1}^{N}\binom{N}{s}X^{N-s}$. It turns out that G is just the number of N-polyominos, known to be α^N with $3.791 < \alpha < 4.649551$ [KR73], approximated to be 4.0625 [CG95].

[26]One of the most difficult tasks that a chemist must ever perform is purifying a chemical product that she desires away from *side products* — chemicals different from her desired product that are created by her synthesis because chemical reactions are not entirely deterministic. If the side products are few in number, small in quantity, or if they have very different chemical structures from her desired product her purification efforts may be successful. Often this is not the case and a particular synthetic route to a compound must be abandoned because no effective purification can be found. Side products are no joke: at best they waste precious starting materials and at worst they have disastrous consequences — the teratogenic effects of the morning-sickness drug thalidomide are attributed to a contaminating side product.

are just a waste of tiles. Intellectually, purification is tantamount to cheating — we can no longer say that we just "self-assembled" an assembly and we must consider the whole system of self-assembly plus purification when calculating the complexity of an assembly.[27]

For these reasons, we limit our theoretical study to tile systems that uniquely produce. We recognize unique production as a notion of efficiency — thus we say that a tile system is **assembly-efficient** for A iff $\mathbf{T}$ uniquely produces A.[28,29,30] Fortunately, this restriction does not appear to weaken the power of tile systems; assembly-efficient tile systems are universal. Via assemblies that simulate 1D cellular automata, Erik Winfree has shown that: for all Turing machines $\mathcal{M}$, a tile system $\mathbf{T}$ can be constructed such that if $\mathcal{M}$ halts with output X, the tile system uniquely produces an assembly with X (written as the labels on tiles) along one edge; if $\mathcal{M}$ does not halt, $Prod(\mathbf{T})$ is not finite. Later, in Theorem 15 we show this result for Turing machines directly.

2.4.3 Correctness of self-assembly programs

Just as for computer programs, the correctness of self-assembly programs must be proven — for the programs given here correctness means that a tile system $\mathbf{T}$ uniquely

[27] An amazing amount of complexity can be hidden in a "purification." Formally, as research in DNA computation has shown, purification is equivalent to an important subset of computation; for example, the complete solution of an NP-complete problem can be embedded in operations that a chemist would consider purification.

[28] We also say that a tile system is **assembly-efficient** for a finite set of assemblies $\mathcal{A}$ iff $\mathbf{T}$ exactly produces $\mathcal{A}$. See Section 2.6 for a few thoughts on exact production.

[29] Depending on experimental circumstances weaker notions of assembly-efficiency could be used. For example one might allow a few additional assemblies in $Term(\mathbf{T})$, perhaps a constant number or a number polynomial in $|A|$. Such definitions may even be practical: in lab we routinely purify an N-nucleotide oligomer of DNA from $O(N)$ side-products using gel-electrophoresis. Using 2D electrophoresis we might hope to purify an N-unit molecule from say $O(N^2)$ side-products. Unless our desired product is very special, however, it seems difficult to purify away an exponential number of side-products.

[30] Actually, our definition of $Prod(\mathbf{T})$ does not exactly match up with our knowledge of real self-assembling systems — real systems may produce more assemblies than we admit and thus our $Prod(\mathbf{T})$ involves some implicit "purification". See Section 2.6 for a discussion.

produces a target assembly A. If it is known that $A \in Term(\mathbf{T})$, which we will see can be decided in time $O(|T||A|)$, unique production of A can be decided[31] in time $O(|T|^{|A|})$ by exhaustively generating all $C \in Prod(\mathbf{T})$ with $|C| \leq |A|$ tiles and for each of them showing $C \leq A$. Such an algorithm is impractical for even small assemblies so we give two algorithms, polynomial in $|T|$ and $|A|$, for deciding unique production.[32] We give a third algorithm, also polynomial in $|T|$ and $|A|$ that verifies the unique production of assemblies with set of special properties.

The first algorithm, due to David Kempe and me, is general: it decides unique production for all tile systems $\mathbf{T}$ and assemblies A in time $O(|A|^2 + |T||A|)$. Unfortunately, it does not capture the idea of translation-invariant unique production; there exist tile systems $\mathbf{T}$ and assemblies A for which the algorithm decides "$\mathbf{T}$ *does not* uniquely produce A" even though all the members of $Term(\mathbf{T})$ are equivalent to A under translation — Figure 2.8a is an example. The second algorithm, derivative of the first, does capture our idea of translation invariant unique production: it decides unique production for all tile systems and *ti*-assemblies $\widetilde{A}$ in time $O(|T||A|^4)$. Interestingly, the algorithm can be adapted to decide whether a finite set of assemblies or *ti*-assemblies is exactly produced (see Section 2.6.7). The third algorithm verifies the unique production of assemblies having certain special properties. These special properties allow us to design assemblies that

[31]Note that we are describing algorithms for deciding $\mathcal{UP}^A$, the set of pairs $\langle \mathbf{T}, A \rangle$ such that $\mathbf{T}$ uniquely produces A. Because, by Theorem 15, we can construct tile systems that uniquely produce iff an arbitrary Turing machine halts, the set $\mathcal{UP}$ of tile systems that uniquely produce is *undecidable*.

[32]Since $|T|$ seems likely to be much smaller than $|A|$, one might ask, "Why carry around $|T|$ in the time complexity?" In fact, later, we show that for some "assembly problems" one must use unique addressing such that $|T| = |A|$. Also, we may run into physical situations where $|T|$ is bigger than $|A|$. For example, we may wish to learn whether an assembly A that is uniquely produced by $\mathbf{T}_1$ (with tile set T_1) is still uniquely produced in a physical system where other tile sets $T_2 \ldots T_n$ are present. Then we need to re-run our algorithm for deciding whether A is uniquely produced with a new tile set $T_1 \cup T_2 \cup \ldots \cup T_n$, possibly much larger than $|A|$.

are uniquely produced. Thus, by design, all of the constructions we give for bounding the program size complexity of squares at $\mathcal{T} = 2$ have these special properties and our third algorithm verifies their unique production very quickly, in time $O(|T| + |A|)$.

2.4.3.1 Unique production of assemblies

We begin with simple algorithms for deciding whether assemblies are produced and terminal and use these as subroutines for the algorithms that decide unique production. For all the algorithms that follow, we assume that A is supplied both in the form of a linked list of tiles t and sites (x, y) so that the tiles of A can be enumerated in time $O(|A|)$ and as a 2D array indexed by these sites so that neighbors of a tile at (x, y) can be determined in constant time. We also assume that a binary array of length $|T|$ indicates whether or not tile t is one of the seed tiles so this can be determined in constant time.

Theorem 2 *$\forall \langle \mathbf{T}, A \rangle$, $\mathbf{T} = \langle T, \mathcal{S}, g, \mathcal{T} \rangle$ is a discrete tile system, g is non-negative and A is a finite assembly: "$A \in Prod(\mathbf{T})$" can be decided in time $O(|A|)$.*

PROOF. The production of A from $S \in \mathcal{S}$ can be decided by a simple greedy algorithm that constructs an assembly $C = A$ from S using A as a reference. It does so quickly by keeping a linked list L of exactly those empty sites where C can grow.

First, check that $A(0,0) = S(0,0)$ for some $S \in \mathcal{S}$; if not, return "NO". Next, initialize $C(0,0)$ with $A(0,0)$ and initialize L with sites (i, j) neighboring $(0,0)$ such that $A(i, j)$ can add to C with a bond strength $\geq \mathcal{T}$. Note that because bond strengths are non-negative, once a site goes onto L it should only be taken off when the site is filled. Also keep a binary 2D-array b of which sites have been added to L. Now attempt to construct A in

$O(|A|)$ steps. At each step: remove the first site (x, y) from L, fill $C(x, y)$ with the tile at $A(x, y)$, and add to L all sites (i, j) neighboring (x, y) such that $A(i, j)$ can add to C *and* (i, j) is not in already L (use b as a reference). This is a constant amount of work per tile added. Halt with "NO" if L becomes empty before A is constructed; with "YES" if A is constructed.

It is clear that the algorithm does not terminate with "YES" unless A is produced. No more than $4|A|$ sites can ever be added to L and at every step one is removed so the algorithm always terminates. But why can't the algorithm return "NO" if A is produced? The reason is again the non-negativity of bond strengths. Assume A is produced but the algorithm returns "NO". Consider C when the algorithm returns "NO". For some assembly sequence producing A, let (x, y) be the first site filled for which $C(x, y) = \textit{empty}$, with $B + \square_t^{(x,y)}$ being the crucial step. Note $B \subseteq C$; then, by non-negativity of bond strengths, a site that can be filled in B can also be filled in C (Lemma 1). Therefore (x, y) should have been in L when the algorithm halted saying "NO". Contradiction. ■

Theorem 3 *$\forall \langle \mathbf{T}, A \rangle$, $T = \langle T, \mathcal{S}, g, \mathcal{T} \rangle$ is a discrete tile system, g is non-negative, and A is a finite assembly: "$A \in Term(\mathbf{T})$" can be decided in time $O(|T||A|)$.*

Proof. $A \in Term(\mathbf{T})$ iff [$A \in Prod(\mathbf{T})$ and A is maximal]. First check if $A \in Prod(\mathbf{T})$ using the algorithm from Theorem 2. Next we check that A is maximal: build a list of open sites in A by running through the list of tiles in A and referencing the array representation of A to find empty neighbors — this takes time $O(|A|)$. If the list is empty, return "YES"; otherwise check to see whether any of the $|T|$ tiles in T can bind to A at an open site. Return "NO" if any tile can bind and "YES" otherwise. The maximum number of open

sites on A can be no more than the $4|A|$ sites neighboring tiles in A.[33] Thus the algorithm takes time $O(|T||A|)$. Correctness follows from the definition of $Term(\mathbf{T})$. ■

In order to generate our first algorithm for deciding whether a tile system uniquely produces, we first determine some necessary and sufficient conditions for unique production. The definition of unique production is a little awkward to use directly. Thus we prove:

Lemma 4 *$\forall \langle \mathbf{T}, A \rangle$ $\mathbf{T} = \langle T, \mathcal{S}, g, \mathcal{T} \rangle$ is a discrete tile system, g is non-negative, and A is an assembly s.t. $A \in Term(\mathbf{T})$: $\mathbf{T}$ uniquely produces A iff there does not exist a $B \in Prod(\mathbf{T})$ such that $B \not\leq A$.*

PROOF. **(if)** Assume not. Then "there does not exist a $B \in Prod(\mathbf{T})$ s.t. $B \not\leq A$" is consistent with "$\mathbf{T}$ does not uniquely produce A". Then either:

1. $\mathbf{T}$ is non-halting. Then either:

   (a) $Term(\mathbf{T})$ is infinite, so there must exist another element $C \in Term(\mathbf{T})$, $C \neq A$. Let $B = C$. Contradiction. OR

   (b) $\mathbf{T}$ is non-haltable. By defn. $\exists C \in Prod(\mathbf{T})$ s.t. $\forall A' \in Term(\mathbf{T}) : C \not\leq A'$. Specifically there must exist a $C \not\leq A$; let $B = C$. Contradiction. OR

2. $Term(\mathbf{T}) \neq \{A\}$. $A \in Term(\mathbf{T})$ so there must exist another element $C \in Term(\mathbf{T})$, $C \neq A$. Let $B = C$. Contradiction.

(only if) Assume not. Then "T uniquely produces A" is consistent with "there exists a $B \in Prod(\mathbf{T})$ such that $B \not\leq A$". Either:

[33] Actually, by induction on the number of tiles, the maximum number of open sites is $2|A| + 2$. The seed has four open sites. The addition of a tile destroys at least one open site and can create no more than three; thus adding a tile adds at most two open sites. The maximum occurs in the case that A is linear.

1. $\exists B' \geq B$ which is terminal. Thus $B' \in Term(\mathbf{T})$, $B' \neq A$ and $Term(\mathbf{T}) \neq \{A\}$. Contradiction. OR

2. $\nexists B' \geq B$ which is terminal. Thus $\mathbf{T}$ is non-halting. Contradiction.

■

How do we determine whether or not a $B \in Prod(\mathbf{T})$ such that $B \not\leq A$ exists? The existence of B implies a type of non-determinism in the assembly process — alternative paths of assembly sequence branching away from the sequences that lead to A, towards B, never to return. We can try to find those places in an assembly sequence for A where the assembly first takes an irrevocable step towards B. We characterize such crucial steps as follows: For all tile system, assembly pairs $\langle \mathbf{T}, A \rangle$ we say that a triple $\langle (x,y), t, C \rangle$ is a **witness for the non-uniqueness of assembly** A where (x,y) is a site, t a tile, and C an assembly iff $A(x,y) \notin \{t, empty\}$, $C \in Prod(\mathbf{T})$, $C \leq A$, and $C' = C + \square_t^{(x,y)} \in Prod(\mathbf{T})$. A trivial but often used consequence of this definition is that if $\langle (x,y), t, C \rangle$ is a witness for the non-uniqueness of assembly A, then $C' \not\leq A$. We are loose with our language and say that an assembly sequence to A "passes through" a witness $\langle (x,y), t, C \rangle$ if C occurs in that assembly sequence. The ideas to remember are (1) that C is an assembly occurring along some assembly sequence for A and (2) that C *is an assembly at which we have a choice* — we could continue to add tiles from A, or we could add a tile t to create an assembly $C' \not\leq A$. Now we prove:

Lemma 5 $\forall \langle \mathbf{T}, A \rangle$ $\mathbf{T} = \langle T, \{S\}, g, \mathcal{T} \rangle$ *is a singly seeded discrete tile system,* g *is non-negative, and* A *is an assembly s.t.* $A \in Term(\mathbf{T})$*:* $\mathbf{T}$ *uniquely produces* A *iff there* does not *exist a witness* $\langle (x,y), t, C \rangle$ *for the non-uniqueness of* A*.*

PROOF. By application of Lemma 4 and negation of the result, it suffices to prove that: There exists a $B \in Prod(\mathbf{T})$ s.t. $B \not\leq A$ iff there exists a witness $\langle (x,y), t, C \rangle$ for the non-uniqueness of A.

(if) Trivial, let $B = C + \square_t^{(x,y)}$. **(only if)** Let C be a maximal assembly such that P =[$C \in Prod(\mathbf{T})$ and $C \leq A$ and $C \leq B$] as shown in Figure 2.5a. (Note that by Lemma 3, C is unique, *i.e.* the meet of A and B). C exists because (i) A is an upper bound for C and (ii) $\mathbf{T}$ is singly seeded, thus $S \leq A$ and $S \leq B$ — if no greater assembly satisfies P, S satisfies P. Clearly $C < B$ since $B \not\leq A$. Thus we can add a tile from B to C: there exists a site (x,y) and a tile t such that $C' = C + \square_t^{(x,y)}$, $C' \in Prod(\mathbf{T})$ and $C' \leq B$. Since $C \leq A$ then $C \subseteq A$ so any tiles that surround (x,y) in C also surround (x,y) in A. If $A(x,y) = empty$ then, by non-negativity, $A + \square_t^{(x,y)}$ could be produced. This would contradict the assumption that $A \in Term(\mathbf{T})$; thus, $A(x,y) \neq empty$. Finally, $A(x,y) \neq t$; otherwise would contradict the assumption that C is a maximal assembly such that $C \leq A$ and $C \leq B$. Thus $\langle (x,y), t, C \rangle$ is a witness for the non-uniqueness of A.

■

Lemma 5 reduces the problem of determining whether or not $\mathbf{T}$ uniquely produces A to the problem of determining whether or not a witness $\langle (x,y), t, C \rangle$ exists. Thus, we would like an algorithm that, given $\langle \mathbf{T}, A \rangle$ as input, attempts to constructs a witness $\langle (x,y), t, C \rangle$ — succeeding iff a witness exists. Unfortunately, in the proof above, C was constructed as the meet of A and B (Figure 2.5a). Without knowledge of B, we know only that C lies along one of possibly exponentially many assembly sequences to A. Fortunately, iff a witness exists, we *can* construct *another* assembly C_{max} (Figure 2.5b) with the same properties as C but with additional properties that constrain the search for a witness:

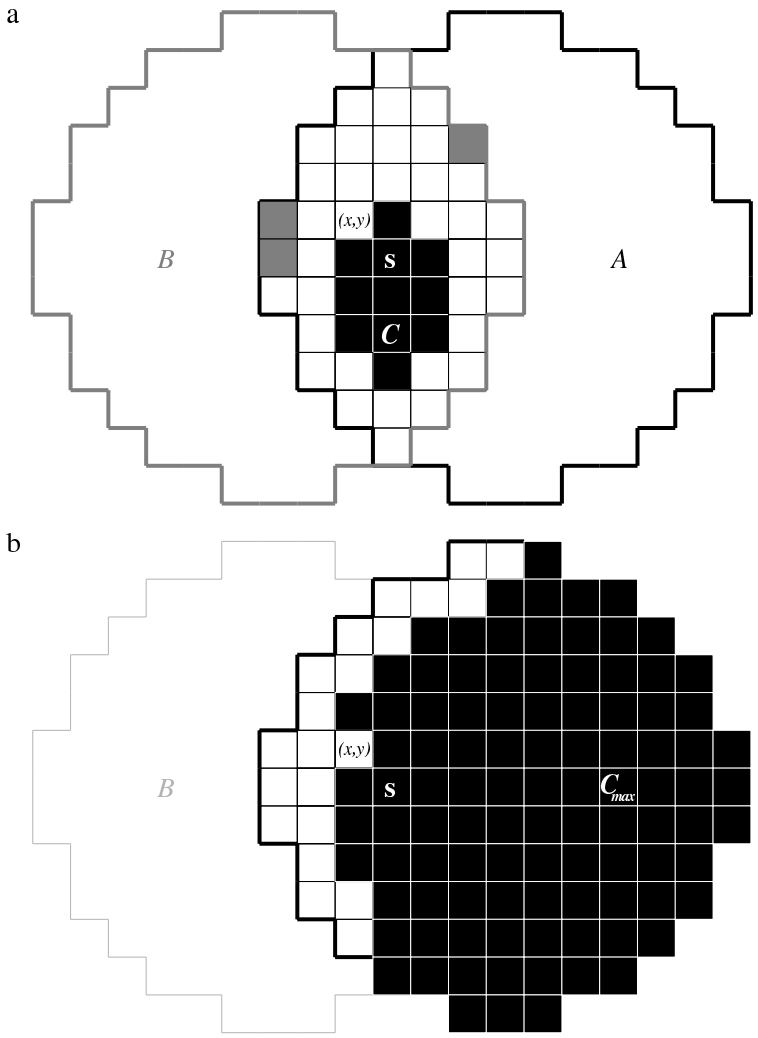


Figure 2.5: Constructing a witness for the non-uniqueness of A. If $A \in Term(\mathbf{T})$ is not uniquely produced, then $B \in Prod(\mathbf{T})$ exists such that $B \not\leq A$. (a) Tiled area depicts the physical overlap of A and B. Gray or black tiles depict $A \cap B$ — positions where A and B are the same. Black tiles depict C, a maximal produced assembly such that $C \leq A$ and $C \leq B$; by Lemma 3 C is unique — the meet of A and B. C may be thought of as the "maximum growable intersection" of A and B. The gray tiles show positions where A and B are coincidentally the same — their addition depends on neighbors that are different in A and B. The site (x, y) is a site where A and B disagree and $B(x, y) = t$ can be added to C to yield $C' \not\leq A$. Thus $\langle (x, y), t, C \rangle$ is a witness for the non-uniqueness of A. (b) B is deemphasized to show that we have no knowledge of it. Now black tiles depict tiles in C_{max}, a maximal produced assembly such that $C \leq A$ and $C_{max}(x, y) = empty$. White tiles depict positions that depend on the addition of $A(x, y)$ to be filled. Notice that $C_{max} \supseteq C$ and that, by non-negativity, the addition of a tile north of (x, y) does not affect the possible addition of t at (x, y). Thus, by non-negativity, $C'_{max} = C_{max} + \square_t^{(x,y)} \in Prod(\mathbf{T})$ so that $C_{max} \not\leq A$ and $\langle (x, y), t, C_{max} \rangle$ is a witness for the non-uniqueness of A. C_{max} is also unique; Theorem 4 gives an algorithm for constructing $\langle (x, y), t, C_{max} \rangle$.

Lemma 6 *$\forall\langle \mathbf{T}, A\rangle$, $\mathbf{T} = \langle T, \mathcal{S}, g, \mathcal{T}\rangle$ is a discrete tile system, g is non-negative, and $A \in Prod(\mathbf{T})$ is a finite assembly: there exists witness $\langle (x,y), t, C\rangle$ for the non-uniqueness of A iff*

1. *C_{max} is the unique maximal assembly such that $C_{max} \in Prod(\mathbf{T})$, $C_{max} \leq A$ and $C_{max}(x,y) = empty$.*

2. *$\langle (x,y), t, C_{max}\rangle$ is a witness for the non-uniqueness of A.*

PROOF. **(if)** Trivial, let $C = C_{max}$. **(only if)** Let C_{max} be a maximal assembly such that $P = [C_{max} \in Prod(\mathbf{T})$, $C_{max} \leq A$ and $C_{max}(x,y) = empty]$ is true. C_{max} exists since (i) a configuration with all the tiles of A but (x,y) empty serves as an upper bound for C_{max} (ii) there exists an $S \in \mathcal{S}$ such that $S \leq A$ and S satisfies P — if no greater assembly satisfying P exists S is a maximal assembly satisfying P.

We now show that C_{max} is unique by contradiction. Assume $\exists D_{max} \neq C_{max}$ and D_{max} is a maximal assembly satisfying P. Let E be a maximal assembly such that $Q = [E \in Prod(\mathbf{T})$, $E \leq C_{max}$ and $E \leq D_{max}]$. E exists since (i) a configuration with all the tiles of A but (x,y) empty serves as an upper bound for E and (ii) S satisfies Q. Clearly, $E < C_{max}$ (otherwise $C_{max} \leq D_{max}$ violating the maximality of C_{max} for P). Thus we can add a tile from C_{max} to E: there exists a nonempty tile u and position (i,j) such that $E + \square_u^{(i,j)} \leq C_{max}$; also note that, critically, $(i,j) \neq (x,y)$ since $C_{max}(x,y) = empty$. Further, $E + \square_u^{(i,j)} \not\leq D_{max}$; otherwise would violate the maximality of E for Q. Thus C_{max} and D_{max} disagree at (i,j) — wherever C_{max} and D_{max} disagree one is empty and the other is not otherwise they could not both assemble to A. Thus D_{max} must be empty at (i,j). Since $E \subseteq D_{max}$, the non-negativity of bond strengths (by Lemma 1

and $D_{max} \in Prod(\mathbf{T})$) means that $D'_{max} = D_{max} + \square_u^{(i,j)}$ must be in $Prod(\mathbf{T})$. Further since $D'_{max} \subseteq A$ and $A \in Prod(\mathbf{T})$, Lemma 2 implies $D'_{max} \leq A$ and since $(x,y) \neq (i,j)$, $D'_{max}(x,y) = empty$; thus D'_{max} violates the maximality of D_{max} for P.

Because $C \in Prod(\mathbf{T})$, $C \leq A$ and $C(x,y) = empty$, and C_{max} is the unique maximal assembly with these properties, $C \leq C_{max} \leq A$. Thus, $C \subseteq C_{max}$ — any tiles that surround (x,y) in C also surround (x,y) in C_{max}. Hence, by non-negativity, t can be added to C_{max} to give $C'_{max} \in Prod(\mathbf{T})$ such that $C'_{max} \not\leq A$; i.e. $\langle (x,y), t, C_{max} \rangle$ is a witness for the non-uniqueness of A. ■

Now it is clear how to find a witness for the non-uniqueness of an assembly — we just have to search for a "maximal" witness $\langle (x,y), t, C_{max} \rangle$. Thus we have:

Theorem 4 *$\forall \langle \mathbf{T}, A \rangle$, $T = \langle T, \mathcal{S}, g, \mathcal{T} \rangle$ is a discrete tile system, g is non-negative, and A is a finite assembly: "$\mathbf{T}$ uniquely produces A" can be decided in time $O(|A|^2 + |T||A|)$.*

PROOF. The correctness of the following algorithm follows from Lemmas 5 and 6. First, check if $|\mathcal{S}| > 1$; if so, return "NO". Second, as per Theorems 2 and 3, check that $A \in Prod(\mathbf{T})$ in $O(|A|)$ steps and $A \in Term(\mathbf{T})$ in $O(|T||A|)$ steps; failing either, return "NO". Next check for a witness for the non-uniqueness of A: for all $|A|$ non-empty sites (x,y) in A, construct the unique maximal produced assembly $C_{max} \leq A$ that is empty at (x,y). Do this using a greedy algorithm similar to that in Theorem 2 (using $O(|A|)$ steps) but hold the assembly *empty* at (x,y). Test the $|T|-1$ tiles distinct from $A(x,y)$ for the possibility of addition at (x,y) — if a tile t can be added at $\mathcal{T}$ to yield $C'_{max} \not\leq A$, return "NO". If for all $|A|$ assemblies C_{max} no tile other than the correct tile $A(x,y)$ can be added, then return "YES".

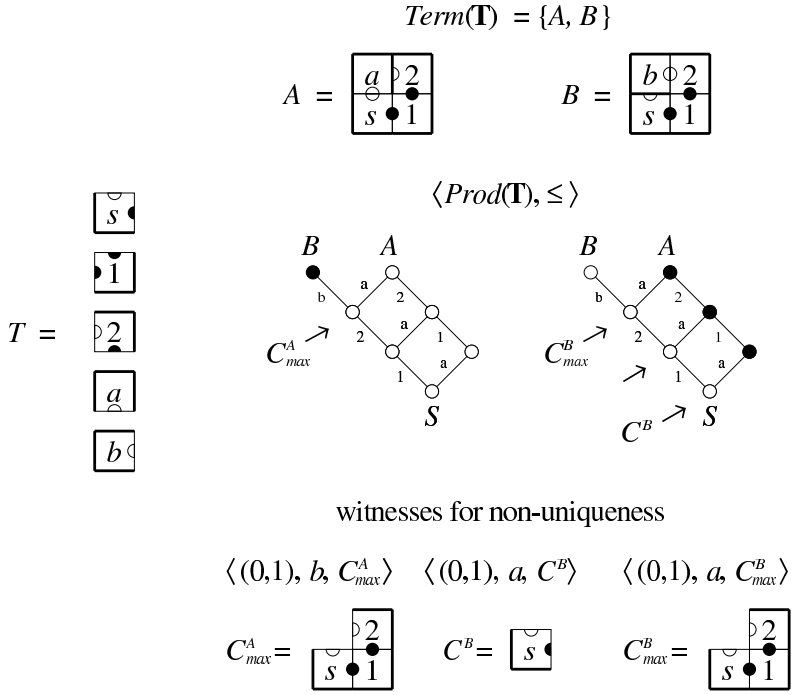


Figure 2.6: Example of a tile system that does not uniquely produce. At left, the set of tiles T. Assemblies A and B are the elements of $Term(\mathbf{T})$ for $\mathcal{T} = 1$. Below each assembly we show a representation of the partially ordered set $\langle Prod(\mathbf{T}), \leq \rangle$, known as a Hasse diagram. Circles, regardless of shading, represent elements of $Prod(\mathbf{T})$; every element of $Prod(\mathbf{T})$ is diagrammed. Edges represent a single step of self-assembly; for two elements $X, Y \in Prod(\mathbf{T})$ there exists an edge extending upwards from X to Y iff $X \rightarrow_{\mathbf{T}} Y$. Each edge is labeled on the left with the tile being added at that step. Each path from the bottom to the top of a Hasse diagram indicates an assembly sequence and corresponds to a linear synthesis tree as in Figure 2.1B. The convergence of two paths means that a single element of $Prod(\mathbf{T})$ can be created by more than one assembly sequence; it *does not* mean that two assemblies have joined as in Figure 2.1A. Black-filled circles indicate elements of $Prod(\mathbf{T})$ that are *not* part of any assembly sequence for the terminal assembly diagrammed directly above. For example, consider the Hasse diagram at right: none of the elements of $Prod(\mathbf{T})$ shaded in black can self-assemble to B. Assemblies that can serve as part of some witness for non-uniqueness are indicated by arrows. Notice that they are indicated by open circles that share edges with black circles; there is one such assembly for A and three such assemblies for B. Below the posets we give sample witnesses the non-uniqueness of A and B; one for A, and two for B (a third for B exists, we do not diagram it). Only the witnesses C^A_{max} and C^B_{max} would be found by the algorithm in Theorem 4.

If the algorithm finds a witness and returns "NO", it is clear that $\mathbf{T}$ does not uniquely produce A. But if the algorithm returns "YES" could it have missed a witness? The algorithm constructs a maximal assembly C_{max} such that $C_{max} \in Prod(\mathbf{T})$, $C_{max} \leq A$ and $C_{max}(x, y) = empty$. This follows from the non-negativity of bond strengths as argued for Theorem 2. Lemma 6 states that such an assembly C_{max} is unique for each particular (x, y); thus our algorithm is forced to construct the unique one. Further, iff a witness $\langle (x, y), t, C \rangle$ exists, $\langle (x, y), t, C_{max} \rangle$ will also be a witness. Thus the algorithm cannot miss detecting a witness if one exists.

Testing for a witness takes $O(|A|(|A| + |T|))$ steps. Thus the total time complexity, including checks to determine that A is produced and terminal, is $O(|A|^2 + |T||A|)$. ■

To understand the algorithm presented above, it may be helpful to consider the execution of the algorithm on the example tile system and assemblies in Figure 2.6. Run on A, the algorithm finds $\langle (0, 1), b, C^A_{max} \rangle$; run on B it finds $\langle (0, 1), a, C^B_{max} \rangle$. Given an assembly, our algorithm tries to find a witness at each non-empty site — why can't one just to run the algorithm for testing production, but add an additional check for a witness for non-uniqueness at each step? Figure 2.6 shows why this simpler algorithm does not work. It *would* detect a witness for the non-uniqueness of B (in particular, $\langle (0, 1), a, C^B \rangle$) since every assembly sequence to B passes through a witness for its non-uniqueness. The algorithm would not *necessarily* have detected a witness for the non-uniqueness of A since not *all* assembly sequences to A pass through a witness for non-uniqueness. This is because the tile b cannot bind until after tile '2' has bound. If the algorithm for production had chosen the assembly sequence that first adds tile a, then tile '1', then tile '2', there never

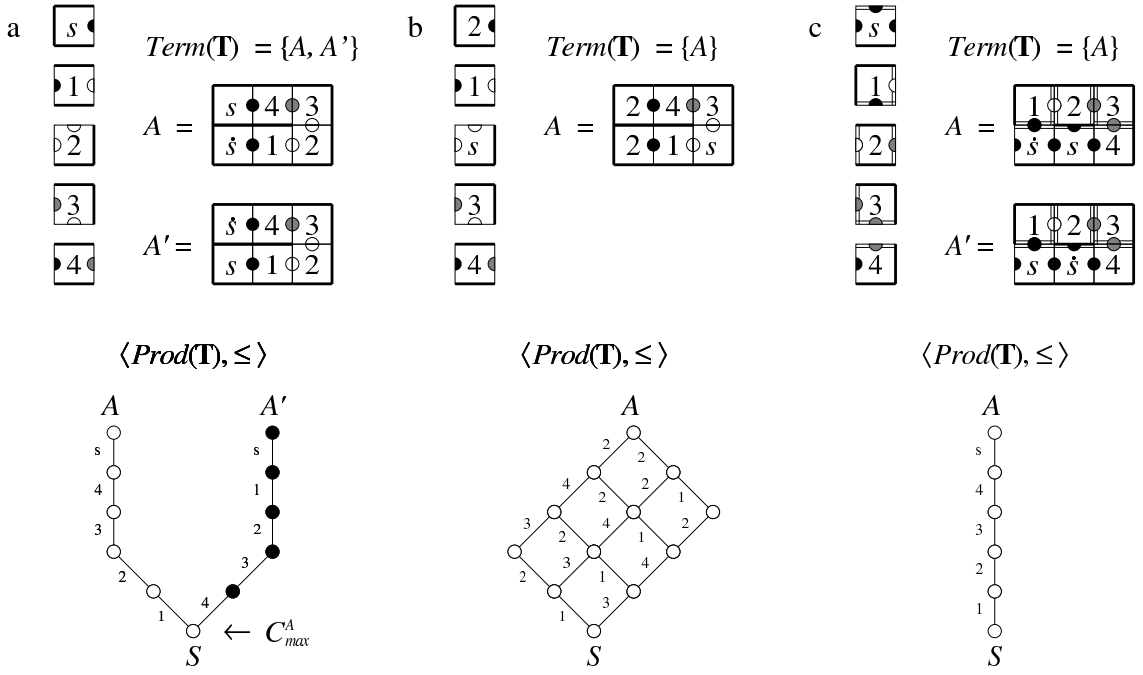


Figure 2.7: Non-uniquely seeded assemblies are not, in general, uniquely produced. Where the location of $(0,0)$ is ambiguous we add a dot to the seed tile ($\dot{s}$) that is at position $(0,0)$. (a) A $\mathcal{T} = 1$ tile system that fails to uniquely produce. Two non-uniquely seeded assemblies, $A \neq A'$ are produced. (b) The same tile system as in (a) *except* the tile that was labeled 2 is now used as the seed tile to make the assembly A uniquely seeded. The tile system uniquely produces the assembly A. This is the type of pathology that arises when considering assemblies rather than *ti*-assemblies. (c) A $\mathcal{T} = 2$ tile system and a non-uniquely seeded assembly A that is uniquely produced. The seed-aligned assembly A', which is equivalent to A under translation, is *not* produced; otherwise A could not be uniquely produced.

would have been an open site at which tile b could bind; the witness would have been missed.

In discussions of the Tile Assembly Model, there is often confusion regarding unique production and multiply-seeded tile-systems or non-uniquely seeded assemblies. First, a multiply-seeded tile system cannot uniquely produce an assembly — a candidate assembly A cannot be produced by any seed tile distinct from $A(0,0)$ and all seed tiles occur in $Prod(\mathbf{T})$. Our algorithm tests for multiply-seeded tile systems and rejects them at the

first step. Second, if A is non-uniquely seeded, a tile system $\mathbf{T}$ that produces A does not, in general, uniquely produce A. The tile system in Figure 2.7a is a typical example; it produces two distinct seed-aligned assemblies $A \neq A'$, both equivalent to $\widetilde{A}$. If either seed-aligned assembly is tested for unique production using our algorithm, a subassembly of the other is found as a witness for non-uniqueness — for example $C^A_{max} = \langle (1,0), \text{`4'}, S \rangle$ is a witness for the non-uniqueness of A.

Interestingly a *very* similar tile system, with the tiles re-labeled so that a different tile is the seed tile and the assembly A is uniquely seeded, *does* uniquely produce a single assembly, as shown in Figure 2.7b. While the behaviour of a *ti*-tile system *can* change when the identity of the seed tile is changed, it happens in more subtle ways. This example is evidence that we really want to be thinking about *ti*-assemblies rather than assemblies.

Stranger still, there do exist tile systems that uniquely produce assemblies with two occurrences of a seed tile. The $\mathcal{T} = 2$ tile system suggested by Matt Cook and depicted in Figure 2.7c is clearly such a tile system. It *uniquely produces* the assembly A with the lower left seed tile aligned to $(0,0)$. It *does not produce* the assembly A' with the lower middle seed tile aligned to $(0,0)$. This example can be generalized to include any number of occurrences of the seed tile.

2.4.3.2 Unique production of *ti*-assemblies

In the real world the identity of a molecule does not change based on its coordinate system. To be consistent with the real world, we would like a point of view that classifies the assemblies A and A' in Figure 2.7a as the same object and allows us to decide whether that object is uniquely produced. Thus, we are really interested in *ti*-assemblies rather

than assemblies. We next develop an algorithm to decide the unique production of *ti*-assemblies in a manner parallel to that used for assemblies. When deciding a question about a *ti*-assembly $\widetilde{A}$ we assume that we are given the assembly A using the same data structures outlined in the previous section. First we show:

Theorem 5 $\forall\langle \mathbf{T}, A\rangle$ $\mathbf{T} = \langle T, \mathcal{S}, g, \mathcal{T}\rangle$ *is a discrete tile system,* g *is non-negative, and* A *is a finite assembly: "*$\widetilde{A} \in \widetilde{Prod(\mathbf{T})}$*" can be decided in time* $O(|A|^2)$.

PROOF. $\widetilde{A} \in \widetilde{Prod(\mathbf{T})}$ iff there exists $A' \in \widetilde{A}$ such that $A' \in Prod(\mathbf{T})$. But if $A' \in Prod(\mathbf{T})$, then $A'(0,0) = S(0,0)$ for some $S \in \mathcal{S}$ — that is, A' is seed-aligned. Thus we need only to see if there exists a seed-aligned $A' \in \widetilde{A}$ that is produced. There are up to $|A|$ seed-aligned assemblies in $\widetilde{A}$. To see if one is produced, run the algorithm from Theorem 2 a total of $|A|$ times, translating the origin once to each non-empty (x, y) in A. ■

We note that it is not sufficient to stop the algorithm and declare $\widetilde{A} \notin \widetilde{Prod(\mathbf{T})}$ after any particular seed-aligned $A' \in \widetilde{A}$ has failed to be produced; a different occurrence of the seed tile might still allow $\widetilde{A}$ to be produced. For example, consider Figure 2.8c. Let $A \in \widetilde{A}$ be the assembly with the bottom right seed tile of $\widetilde{A}$ aligned to $(0,0)$. Let $A' \in \widetilde{A}$ be the assembly such that $A'(0,0) = A(0,1)$. A is produced but A' is not.

We also have:

Theorem 6 $\forall\langle \mathbf{T}, A\rangle$, $\mathbf{T} = \langle T, \mathcal{S}, g, \mathcal{T}\rangle$ *is a discrete tile system,* g *is non-negative, and* A *is an assembly s.t.* $\widetilde{A} \in \widetilde{Prod(\mathbf{T})}$: "$\widetilde{A} \in \widetilde{Term(\mathbf{T})}$*" can be decided in time* $O(|T||A|)$.

PROOF. $\widetilde{A} \in \widetilde{Term(\mathbf{T})}$ iff there exists an $A' \in \widetilde{A}$ such that $A' \in Term(\mathbf{T})$. $A' \in Term(\mathbf{T})$ iff $A' \in Prod(\mathbf{T})$ and A' is maximal. Our given A is produced by assumption.

Notice that, for all $A' \in \widetilde{A}$, [A' is maximal iff for all $A'' \in \widetilde{A}$: A'' is maximal] because maximality is a coordinate system independent property. Thus we just need to check that A is maximal, as in Theorem 3. ■

It is clear from Figure 2.7a that to decide the unique production of a *ti*-assembly $\widetilde{A}$, it is not sufficient to decide the unique production of any *single* seed-aligned assembly in $\widetilde{A}$. Next, we show that we can reduce the problem of deciding the unique production of $\widetilde{A}$ to the problem of determining whether a special subset of its seed-aligned assemblies are *exactly produced*; in particular, the set of seed-aligned assemblies in $\widetilde{A}$ that are *produced*.

Lemma 7 $\forall \langle \mathbf{T}, A \rangle$, $\mathbf{T} = \langle T, \mathcal{S}, g, \mathcal{T} \rangle$ *a discrete tile system, g is non-negative, and A is a finite assembly:* $\mathbf{T}$ *uniquely produces* $\widetilde{A}$ *iff* $\mathbf{T}$ *exactly produces* $\mathcal{A} = \{A' : A' \in \widetilde{A}$, $A' \in Prod(\mathbf{T})$, *and* A' *is seed-aligned*$\}$.

PROOF. **(if)** By defn. of exact production of $\mathcal{A}$, $\mathbf{T}$ is halting. $Term(\mathbf{T}) = \mathcal{A}$ and all $A' \in \mathcal{A}$ are also in $\widetilde{A}$; therefore $Term(\mathbf{T}) = \{\widetilde{A}\}$.

(only if) Assume not. By defn. of unique production of $\widetilde{A}$, $\mathbf{T}$ is halting. Therefore it must be the case that $Term(\mathbf{T}) \neq \mathcal{A}$. Then either:

1. $\exists B \in Term(\mathbf{T})$ s.t. $B \notin \mathcal{A}$. Thus $B \in Prod(\mathbf{T})$. Also $B \in \widetilde{A}$; otherwise $Term(\mathbf{T}) \neq \widetilde{A}$. The only members of $\widetilde{A}$ that can be in $Prod(\mathbf{T})$ are seed-aligned $A' \in \widetilde{A}$. But then $B \in \mathcal{A}$. Contradiction. OR

2. $\exists A' \in \mathcal{A}$ s.t. $A \notin Term(\mathbf{T})$. Then, either $A' \in Prod(\mathbf{T})$ or $A' \notin Prod(\mathbf{T})$. By definition of $\mathcal{A}$, $A' \in Prod(\mathbf{T})$. But then no members of the equivalence class $\widetilde{A}$ are terminal and $\widetilde{Term(\mathbf{T})} \neq \{\widetilde{A}\}$. Contradiction.

■

As before, in order to generate an algorithm, we find necessary and sufficient conditions for a tile system $\mathbf{T}$ to exactly produce a set of assemblies $\mathcal{A}$. As before, the definition of exactly produces is a little unwieldy so we show:

Lemma 8 *$\forall \langle \mathbf{T}, \mathcal{A} \rangle$, $\mathbf{T} = \langle T, \mathcal{S}, g, \mathcal{T} \rangle$ is a discrete tile system, g is non-negative, and $\mathcal{A}$ is a set of assemblies s.t. $\mathcal{A} \subseteq Term(\mathbf{T}) : \mathbf{T}$ exactly produces $\mathcal{A}$ iff there does not exist a $B \in Prod(\mathbf{T})$ such that $\forall A \in \mathcal{A} : B \not\leq A$.*

PROOF. Similar to the proof of Lemma 4. ■

Our previous idea to use a witness for non-uniqueness still seems capable of detecting B, but if we have a witness for the non-uniqueness of some $A \in \mathcal{A}$ we have to make sure that the putative B we are detecting isn't just another member of $\mathcal{A}$. Thus, we say that $\langle (x, y), t, C, A \rangle$ is a **witness for the non-exactness of a set of assemblies** $\mathcal{A}$ iff $A \in \mathcal{A}$ and $\langle (x, y), t, C \rangle$ is a witness for the non-uniqueness of A such that for all $A' \in \mathcal{A}$: $C + \square_t^{(x,y)} \not\subseteq A'$. For fun, we say that a witness for the non-uniqueness of $A \in \mathcal{A}$ is **unreliable** concerning the non-exactness of $\mathcal{A}$ iff there exists an $A' \in \mathcal{A}$ such that $C + \square_t^{(x,y)} \subseteq A'$. The idea behind these definitions is that a witness for the non-uniqueness of $A \in \mathcal{A}$ disproves the exactness of $\mathcal{A}$ *unless* some valid assembly sequence for another $A' \in \mathcal{A}$ passes through the witness — that is, the witness is unreliable. Thus, parallel to Lemma 5 we prove:

Lemma 9 *$\forall \langle \mathbf{T}, \mathcal{A} \rangle$, $\mathbf{T} = \langle T, \mathcal{S}, g, \mathcal{T} \rangle$ is a discrete tile system, g is non-negative, and $\mathcal{A}$ is a set of assemblies s.t. $\mathcal{A} \subseteq Term(\mathbf{T})$: $\mathbf{T}$ exactly produces $\mathcal{A}$ iff:*

1. *there does not exist a seed tile s such that $\forall A \in \mathcal{A} : A(0,0) \neq s$. AND*

2. *there does not exist a witness $\langle (x,y), t, C, A \rangle$ for the non-exactness of $\mathcal{A}$.*

PROOF. By the application of Lemma 8 and negation of the result, it suffices to prove that: there exists a $B \in Prod(\mathbf{T})$ s.t. $\forall A \in \mathcal{A} : B \not\leq A$ iff it is the case that [not (1) or not (2)].

(if) Depending on which of [not (1)] or [not (2)] is true, let B be the assembly:

$\square_s^{(0,0)}$. This assembly is produced since s is a seed tile, and $\forall A \in \mathcal{A} : \square_s^{(0,0)} \not\leq A$.

$C + \square_t^{(x,y)}$. By the defn. of a witness for the non-uniqueness of A, $C + \square_t^{(x,y)}$ is produced. By the defn. of a witness for the non-exactness of $\mathcal{A}$ we know that for all $A' \in \mathcal{A}$: $C + \square_t^{(x,y)} \not\subseteq A'$ which implies $C + \square_t^{(x,y)} \not\leq A'$.

(only if) Assume not. Let (0) be the statement: [there exists a $B \in Prod(\mathbf{T})$ s.t. $\forall A \in \mathcal{A} : B \not\leq A$]. Then (0) must be consistent with [(1) and (2)] being true.

By (0) B is produced; thus, there exists a seed tile s such that $S = \square_s^{(0,0)} \leq B$. Let $\mathcal{A}'$ be the set of all assemblies $A' \in \mathcal{A}$ such that $A'(0,0) = s$ — that is, assemblies that are share a seed tile at $(0,0)$ with B. By assumption of (1), $\mathcal{A}'$ is not empty. Consider the assembly sequence $S \rightarrow_{\mathbf{T}} B_1 \rightarrow_{\mathbf{T}} B_2 \rightarrow_{\mathbf{T}} \ldots \rightarrow_{\mathbf{T}} B_{|B|} = B$. Since (0) requires $\forall A \in \mathcal{A} : B \not\leq A$, there is some crucial step in the assembly sequence before which at least one $A' \in \mathcal{A}'$ still agrees with B — $B_k \leq A'$ — and after which all $A'' \in \mathcal{A}'$ no longer agrees with B — $B_{k+1} = B_k + \square_t^{(x,y)} \not\leq A''$. Thus we have a witness $\langle (x,y), t, B_k \rangle$ for the non-uniqueness of A'. For statement (2) to be true, then, it must be the case that the witness is unreliable — there exists an $\dot{A} \in \mathcal{A}$ such that $B_{k+1} \subseteq \dot{A}$. Since B_{k+1} and $\dot{A}$ are produced, by Lemma 2, $B_{k+1} \leq \dot{A}$. Contradiction. ■

Again, given $\langle \mathbf{T}, \mathcal{A} \rangle$ we would like a simple algorithm to construct $\langle (x,y), t, C, A \rangle$ iff $\mathcal{A}$ is not exactly produced, but again, without knowledge of B there are many possible assemblies C to test. Thus, parallel to Lemma 6 we prove:

Lemma 10 *$\forall \langle \mathbf{T}, \mathcal{A} \rangle$, $\mathbf{T} = \langle T, \mathcal{S}, g, \mathcal{T} \rangle$ a discrete tile system, g is non-negative, and $\mathcal{A}$ is a finite set of finite assemblies s.t. $\mathcal{A} \subseteq Prod(\mathbf{T})$: there exists witness $\langle (x,y), t, C, A \rangle$ for the non-exactness of $\mathcal{A}$ iff*

1. *C_{max} is the unique maximal assembly such that $C_{max} \in Prod(\mathbf{T})$, $C_{max} \leq A$ and $C_{max}(x,y) = empty$.*
2. *$\langle (x,y), t, C_{max}, A \rangle$ is a witness for the non-exactness of $\mathcal{A}$.*

PROOF. **(if)** Trivial, let $C = C_{max}$. **(only if)** Argue as for Lemma 6. Observe additionally that since $\forall A' \in \mathcal{A} : C + \square_t^{(x,y)} \not\subseteq A'$, and $C_{max} \supseteq C$ we have $\forall A' \in \mathcal{A} : C_{max} + \square_t^{(x,y)} \not\subseteq A'$. That is, the extra tiles present in C_{max} cannot make $C_{max} + \square_t^{(x,y)} \subseteq A'$ for some A' if $C + \square_t^{(x,y)} \not\subseteq A'$ already. ■

Our algorithm immediately follows:

Theorem 7 *$\forall \langle \mathbf{T}, A \rangle$, $\mathbf{T} = \langle T, \mathcal{S}, g, \mathcal{T} \rangle$ is a discrete tile system, g is non-negative, and A is a finite assembly: "$\mathbf{T}$ uniquely produces $\widetilde{A}$" can be decided in time $O(|T||A|^4)$.*

PROOF. The algorithm is analogous to that in Theorem 4 and correctness follows from Lemmas 7, 9, and 10. First, check that A is in $Term(\mathbf{T})$ in $O(|T||A|)$ steps; if not, return "NO". Next, generate the set $\mathcal{A}$: for all seed-aligned assemblies A' in equivalence class $\widetilde{A}$ test whether A' is produced using the algorithm from Theorem 2; if $A' \in Prod(\mathbf{T})$ include

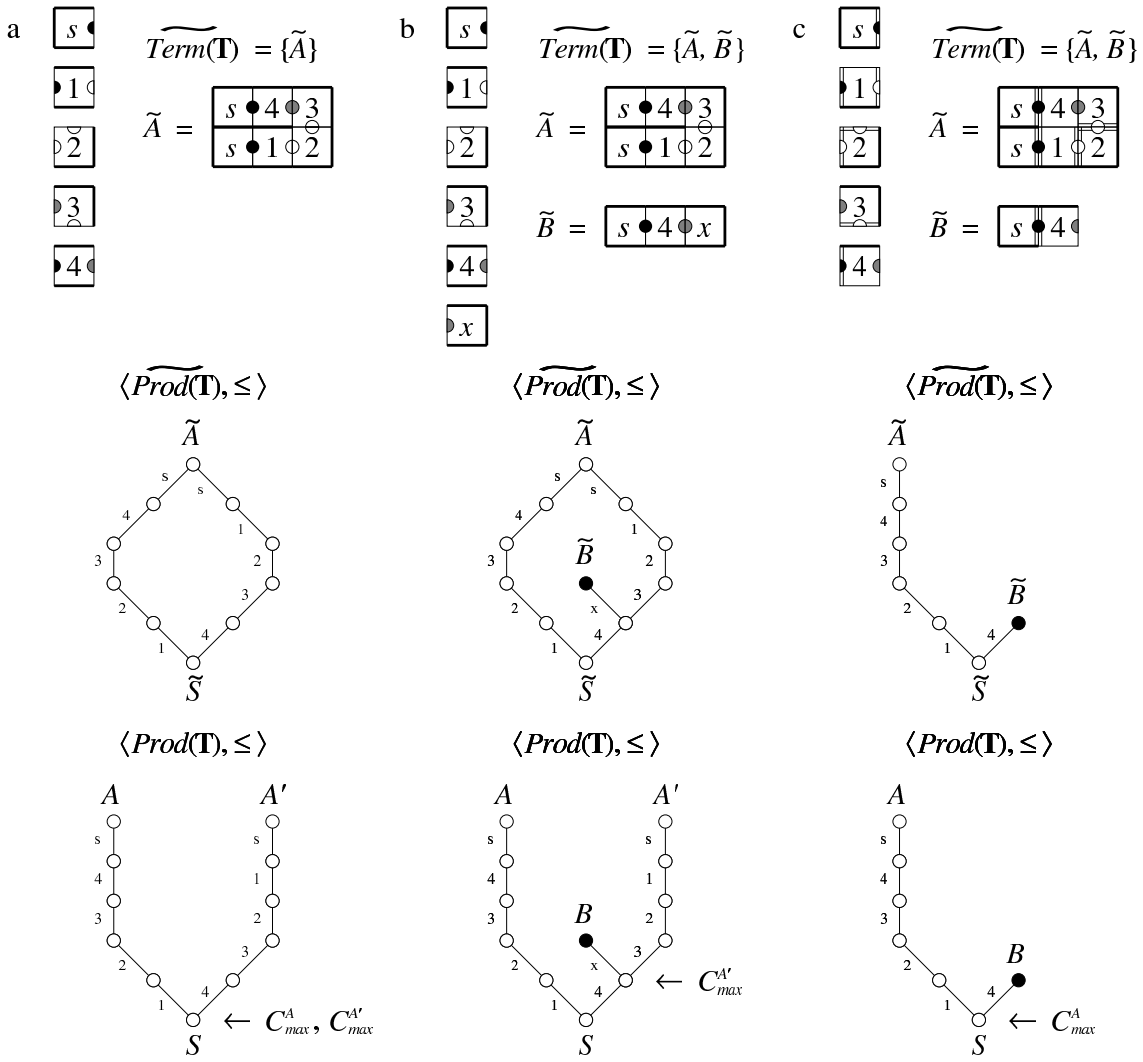


Figure 2.8: Comparison of $Prod(\mathbf{T})$ and $\widetilde{Prod}(\mathbf{T})$ for tile systems with two instances of the same seed tile. For each $\widetilde{A}$, let $A \in \widetilde{A}$ be the assembly with the bottom right seed tile of $\widetilde{A}$ aligned to $(0,0)$ and let $A' \in \widetilde{A}$ be the assembly such that $A'(0,0) = A(0,1)$. (a) A $\mathcal{T} = 1$ tile system with two instances of the same seed tile uniquely produces $\widetilde{A}$. (b) A variant of (a) that obviously fails to uniquely produce $\widetilde{A}$. (c) More surprising, a $\mathcal{T} = 2$ tile system with *exactly the same labels* as in (a) but with a different strength function *does not* uniquely produce $\widetilde{A}$.

it in $\mathcal{A}$. If all tiles in A are seed tiles, this could take time $O(|A|^2)$. Next verify condition (1) of Lemma 9, that for all seed tiles s at least one element of $A' \in \mathcal{A}$ has $A'(0,0) = s$; if not return "NO". This check could take $O(|T||A|)$ steps. Next verify condition (2) of Lemma 9: for all assemblies A' in $\mathcal{A}$ check for a witness for the non-uniqueness of A' as in Theorem 4. If a witness $\langle (x,y), t, C \rangle$ for the non-uniqueness of A' is found, make sure the witness isn't unreliable: Compare $C' = C + \square_t^{(x,y)}$ to all $A'' \in \mathcal{A}$ and verify that $C' \not\subseteq A''$; if so the witness is a reliable witness for the non-exactness of $\mathcal{A}$ — return "NO". If, for all A' no reliable witness exists, return "YES".

The time complexity to find a witness for non-uniqueness is $O(|A|(|A| + |T|))$. Performing the search for all $A' \in \mathcal{A}$ adds a factor of $O(\widetilde{A})$ up front. Comparing C' to all assemblies $A'' \in \mathcal{A}$ inserts an extra $O(|A|^2)$ steps into the inner loop of the algorithm. These additions mean that the time required to check for a witness for non-exactness is $O(|A|^2(|A| + |T||A|^2))$. Thus, including the time it takes to verify that $\widetilde{A} \in \widetilde{Term(\mathbf{T})}$, build the set $\mathcal{A}$, and check for seed tiles, the unique production of $\widetilde{A}$ can be decided in $O(|T||A|^4)$ steps[34]. ■

Again, it may be helpful to consider the execution of the algorithm on several tile system/*ti*-assembly pairs:

1. Figure 2.8a. We saw earlier that this $\mathcal{T} = 1$ tile system *did not* uniquely produce an assembly; the tile system *does* uniquely produce the *ti*-assembly $\widetilde{A}$. To decide this example, the algorithm decides that the tile system exactly produces $\mathcal{A} = \{A, A'\}$. It finds $\langle (1,0), \text{'4'}, S \rangle$ and $\langle (1,0), \text{'1'}, S \rangle$ as witnesses for the non-uniqueness of A

[34] We believe that pre-computing meets for all pairs of seed-aligned assemblies (with the same seed tile) or memorizing unreliable witnesses might lead to an $O(|T||A|^3)$ algorithm — our intent here however was merely to show that deciding "$\mathbf{T}$ uniquely produces $\widetilde{A}$" is in polynomial time.

and A' but dismisses them as unreliable witnesses for the non-exactness of $\mathcal{A}$ since $S+\square^{(1,0)}_{\text{`4'}} \subseteq A'$ and $S+\square^{(1,0)}_{\text{`1'}} \subseteq A$. Clearly this example can be extended to uniquely produced assemblies with an arbitrary number of occurrences of the same seed tile.

2. Figure 2.8a, again. This time, consider all five tiles in T to be seeds; now $\mathbf{T}$ is multiply seeded (also unseeded) but it still uniquely produces $\widetilde{A}$. The algorithm decides this by deciding that $\mathcal{A}$, now with six elements, is exactly produced.

3. Figure 2.8b. This variant of Figure 2.8a does not uniquely produce. The algorithm decides this by deciding that $\mathbf{T}$ does not exactly produce $\mathcal{A} = \{A, A'\}$. This time the witness $\langle (2,0), x, C^{A'}_{max} \rangle$ for the non-uniqueness of A' is found to be a reliable witness for non-exactness because $C^{A'}_{max} + \square^{(2,0)}_{x} \not\subseteq A$. ($C^{A'}_{max} = S + \square^{(1,0)}_{\text{`4'}}$).

4. Figure 2.8c. This $\mathcal{T} = 2$ tile system has *exactly the same labels* as the tile system in Figure 2.8a but it has a different strength function. Surprisingly, it *does not* uniquely produce $\widetilde{A}$. The algorithm decides this example by deciding that the corresponding the tile system does not exactly produce $\mathcal{A} = \{A\}$. The witness for the non-uniqueness of A $\langle (1,0), \text{`4'}, S \rangle$ is found to be a *reliable* witness for non-exactness. This is correct even though $S + \square^{(1,0)}_{\text{`4'}} \subseteq A'$ because $A' \notin \mathcal{A}$ because $A' \notin Prod(\mathbf{T})$. This example demonstrates that Lemma 2 *does not hold* for *ti*-assemblies: Clearly $\widetilde{B}, \widetilde{A} \in \widetilde{Prod(\mathbf{T})}$ and $\widetilde{B} \subseteq \widetilde{A}$ but because tile '4' connects to tiles '3' and '1' by weak strength-1 bonds, $\widetilde{B} \not\trianglelefteq \widetilde{A}$.

5. Figure 2.7c. This tile system uniquely produces $\widetilde{A}$. The algorithm decides this by deciding that $\mathbf{T}$ exactly produces $\mathcal{A} = \{A\}$. No witnesses for the non-uniqueness of A are ever found.

In examining the algorithm one might wonder why we define $\mathcal{A}$ to be just those seed-aligned assemblies $A \in \widetilde{A}$ that are in $Prod(\mathbf{T})$ rather than *all* seed-aligned $A \in \widetilde{A}$. Example 4 above makes it clear: if we had included all seed-aligned assemblies then $A' \notin Prod(\mathbf{T})$ would have been included, $\langle (1,0), \text{`4'}, C^A_{max} \rangle$ would have been found unreliable, and $\widetilde{A}$ would have been misclassified as uniquely produced. To fix this, one might be tempted to conclude "Ah, a *ti*-assembly is uniquely produced iff *all* its seed-aligned assemblies are produced, so I can exclude Example 4 by testing for this ahead of time." But this is wrong, as evidenced by Example 5 which is uniquely produced even though it has a seed-aligned assembly A' that is not in $Prod(\mathbf{T})$.

2.4.3.3 Design of uniquely produced *ti*-assemblies.

First, we note that if a tile system uniquely produces an assembly A, that tile system will also uniquely produce the *ti*-assembly $\widetilde{A}$. Thus, for the purposes of design, we will only consider tile systems that uniquely produce assemblies and so our proofs make no mention of *ti*-assemblies. We do this at the risk of throwing away some interesting tile systems that uniquely produce a *ti*-assembly $\widetilde{A}$ but not the assembly A. This does not seem like a big loss — we have no deep insight into designing such tile systems anyway.

Our general algorithms for deciding unique production are easy to describe. Further, they can decide the correctness of a *particular* uniquely producing self-assembly program relatively quickly. Unfortunately, (i) they cannot be applied to infinite families of self-assembly programs and (ii) they give us little insight as to how one goes about designing a uniquely producing self-assembly program in the first place. What we would like are

one or more simple properties that, if built into a self-assembly program during design, guarantee that the program uniquely produces.

The first simple property that suggests itself is unique addressing — with a few other simple properties unique addressing guarantees unique production:

Theorem 8 *$\forall\langle\mathbf{T}, A\rangle$, $\mathbf{T} = \langle T, \mathcal{S}, g, \mathcal{T}\rangle$ is a discrete tile system, A is an assembly s.t. (i) T singly seeded, (ii) $\mathcal{T} = 1$, (iii) A is a full (iv) A is uniquely addressed (v) T is the set of tiles in A (vi) all neighboring pairs of tiles in A share a unique label of strength 1, and (vii) where tiles border empty tiles they have null binding domains: $\mathbf{T}$ uniquely produces A.*

PROOF. First, we show that A is produced by induction. Assume that for all $1 \leq n \leq |A|$ there exists a produced assembly C_n such that $|C_n| = n$ and $C \subseteq A$. Base case: A is uniquely addressed and T is the set of tiles in A so A contains the seed tile s and $S \subseteq A$. Inductive step: Consider a produced assembly C_n with n tiles such that $C \subseteq A$. A is a 1-stable assembly so $B + C = A$ and B connects to C via at least one strength-1 bond between a tile in B and a tile in C. The tile in B can be added to C_n to form C_{n+1}, also produced, with $|C_{n+1}| = n + 1$ and $C \subseteq A$.

Next we show that A is uniquely produced. Assume not. Since tile edges bordering *empty* tiles have null interactions, A is terminal. Then by Lemma 5 there exists a witness $\langle (x, y), t, C\rangle$ for the non-uniqueness of A such that $C' = C + \square_t^{(x,y)}$, $C' \in \mathit{Prod}(\mathbf{T})$, $C \leq A$, $A(x, y) \notin \{t, \mathit{empty}\}$. Say that when t is added to C to form C', t binds neighbor $C(i, j)$ by some label σ. Since $C \leq A$ and A is full, then $A(x, y)$ must bind the same tile $C(i, j) = A(i, j)$ via the same label σ. Tile t must occur exactly once in A because $|A| = |T|$ and it cannot occur at $A(x, y)$. Thus the label σ occurs twice in A. Contradiction. ■

We remark that properties (iv) and (v) imply that the assembly is uniquely seeded. With (i) this implies, by Theorem 1, that tile systems designed with properties (i)–(vii) will have the nice property that $Prod(\mathbf{T})$ is isomorphic to $\widetilde{Prod}(\mathbf{T})$. But we have already decided that uniquely addressed assemblies are too tile-inefficient to consider.

What we would *really* like are properties guaranteeing unique production that give us flexibility to recycle tiles. The counter tiles, although they produce an infinite pattern, give some guidance: consider the succession of produced assemblies diagrammed in Figure 2.4. Call the tiles labeled "R" and "B" **border tiles**; call the tiles labeled with numbers **rule tiles**. As assembly proceeds we observe several invariants: (i) only north and west edges of tiles have non-zero strength — growth occurs only on the northwest "front" of the assembly. (ii) if tile can bind at a site, then there is a unique tile that can bind there — that is, growth is deterministic. (iii) in any row there is only one "R" border tile and in any one column there is only one "L" border tile — that is, each row or column makes only one strong bond to a neighboring row or column. For the counter tiles, at $\mathcal{T} = 2$, induction on the size of assemblies using these invariants is sufficient to show that the pattern produced by the counter tiles is unique.

It turns out that invariant (ii) — the "deterministic" property[35] — and invariant (iii) — the "row-column" property — can be used to show that more general sorts of assemblies, with up to four diagonal growth fronts are uniquely produced. The deterministic property and the row-column property are easy features to build into a self-assembly program and so they greatly aid the process of design. To show they guarantee unique production (in

[35]It may seem obvious that the deterministic property is necessary for unique production — this is correct for "translation sensitive" assemblies. But for translation-invariant unique production this is not the case, as is evidenced by Figure 2.8a. The tile that binds to the east of the seed tile is not deterministic — it can be tile 1 or tile 4. Yet the *ti*-assembly $\widetilde{A}$ is still uniquely produced.

conjunction with other simple properties like $\mathcal{T} = 2$) we need to define and explore them more formally.

We begin by classifying the tiles in an assembly according to their position with respect to the seed tile and other tiles in the assembly. It turns out that this classification has greater meaning — it indicates which bonds are formed when a tile is added to the assembly; equivalently it indicate the direction in which the assembly grows. Our intent is that if a tile attaches by a single strength-2 bond we will classify it by one of the four cardinal directions {N, E, S, W}; if the tile attaches by a pair of adjacent strength-1 bonds we will classify it by one of the directions {NE, SE, SW, NW}. The main idea is that if an assembly has the row-column and deterministic properties, *any assembly sequence* for the assembly *will attach tiles as classified* by their directions, and further the assembly will be uniquely produced.

Consider a tile $t = (\sigma_N, \sigma_E, \sigma_S, \sigma_W)$ at (x, y) in a uniquely seeded assembly A, where the seed tile is at $(0, 0)$. We say $A(x, y)$ is an **N-tile** if $y > 0$ and $\hat{g}(\sigma_S) \geq 2$ and $A(x, y)$ binds $A(S(x, y))$. That is, N-tiles lie north of the seed tile, and bind on their south via strength-2 or greater bonds. **E-tiles, S-tiles,** and **W-tiles** are defined similarly; they bind via strength-2 bonds on their west, north and east, respectively. The "L" and "R" border tiles for the binary counter are, for example, W and N-tiles, respectively (Figure 2.4). Observe that in any assembly sequence resulting in A, the first tile in row $y > 0$ must have been an N-tile.

We say that an assembly A has the **row-column (RC)** property iff: it is a full, uniquely seeded assembly, each row north of the seed tile has exactly one N-tile, each row

south of the seed tile has exactly one S-tile, each column east of the seed tile has exactly one E-tile, and each column west of the seed tile has exactly one W-tile.

Consider a tile $t = (\sigma_N, \sigma_E, \sigma_S, \sigma_W)$ at (x, y) in an RC assembly A, where the seed tile is at $(0,0)$, and t is not an N-, E-, S-, or W-tile. We say t is a **NE-tile** in A if either (1) t is directly east of an N-tile and t is not directly south of an E-tile, or (2) t is directly north of an E-tile and t is not directly west of an N-tile. We similarly define **SE-tiles, NW-tiles,** and **SW-tiles.** The rule tiles for the binary counter are, for example, NW-tiles (Figure 2.4).

Suppose $A \in Prod(\mathbf{T})$ where A is an RC assembly and $\mathbf{T} = \langle T, \{S\}, g, 2\rangle$. Then it is straightforward to show that every tile in A is exclusively either the seed tile or an N-, E-, S-, W-, NE-, NW-, SE-, or SW-tile. Figure 2.10 shows such a labeling for an RC square.

Lemma 11 *$\forall\langle \mathbf{T}, A\rangle$, $\mathbf{T} = \langle \mathcal{T}, \{S\}, g, 2\rangle$ is a discrete tile system and A is an RC assembly: every tile in A is classified as exactly one of {seed, N-, E-, S-, W-, NE-, NW-, SE-, SW-}.*

PROOF. We use the notation xX where $x = f(D)$ such that f returns the lowercase first letter of a cardinal direction D and $X \in \{N, E, S, W\}$ to denote the description "directly $f^{-1}(x)$ of an X-tile". For example, "t is nE" should be read: "t is directly north of an E-tile".

First, all tiles can be classified. Consider tile t at (x, y). If t is at $(0,0)$ it is the seed tile. Otherwise, if t satisfies the definition of an N-, E-, S-, or W- tile, it is classified as such. Otherwise either:

1. t is in the seed row; $y = 0$. Either t is east of the seed tile or, symmetrically, west of the seed tile. Assume it is east of the seed tile. Then, because A is RC, and t is not

an E-tile, t is either nE, or symmetrically, sE. Say t is nE. Because $y = 0$, t is not wN. t is (nE and not wN); thus t is an NE-tile.

2. t is north of the seed tile. Then either (i) t is in the seed column; $x = 0$ or (ii) t is east of the seed tile or (iii) t is west of the seed tile. Case (i) is a rotation of case (1) above and thus t is classifiable in Case (i). Cases (ii) and (iii) are symmetric. Thus we assume (ii) t is east of the seed tile. Then because A is RC, and t is not an E-tile or N-tile, t is either: (nE and eN) $\Longrightarrow$ (nE and not wN) $\Longrightarrow$ t is an NE-tile; or (nE and wN) $\Longrightarrow$ (wN and not sW) $\Longrightarrow$ t is an NW-tile; or (sE and eN) $\Longrightarrow$ (sE and not wS) $\Longrightarrow$ t is an SE-tile; or (sE and wN) $\Longrightarrow$ (wN and not sW) $\Longrightarrow$ t is an NW-tile.

3. t is south of the seed tile. Same as Case (2) by symmetry.

Next we show that a tile t cannot be be classified in two ways at once. Assume not. By our definitions, t cannot be both the seed tile and an N-tile, or both the seed tile and an NE-tile. (For each rotationally inequivalent case we discuss only a single rotation.) Likewise, by our definition, t cannot be both an N-tile and an S-tile, both an N-tile and a NE-tile, or both an N-tile and an SW-tile. There are now three non-trivial cases to consider:

Case 1: t is both an N-tile and an E-tile. Refer to Figure 2.9. Then t is both north of the seed tile (in the region indicated by the dashed line) and east of the seed tile (in the region indicated by the dotted line). By the definition of RC, A is produced. For all assembly sequences to A there exists a first tile u outside the gray region. u is either an N- or an E-tile, violating our assumption that A is RC. Contradiction.

Case 2: t is both an NW-tile and an NE-tile. Thus t is [(nW and not eN) or (wN and not sW)] and [(eN and not sE) or (nE and not wN)]. Thus there are four cases:

1. (nW and not eN) and (eN and not sE). Contradiction.
2. (nW and not eN) and (nE and not wN). Thus a W-tile and an E-tile occur in the same column. Contradiction.
3. (wN and not sW) and (eN and not sE). Thus there are two N-tiles in a single row, one west and one east of t. Contradiction.
4. (wN and not sW) and (nE and not wN). Contradiction.

Case 3: t is both an SW-tile and an NE-tile. Refer to Figure 2.9. t is (i) [(sW and not eS) or (wS and not nW)] as denoted by the dashed line (Figure 2.9) and (ii) [(eN and not sE) or (nE and not wN)] as denoted by the dotted line. Without loss of generality, assume that t is in the "northwest" intersection of these two regions. Thus t is (sW and not eS) and (eN and not sE); thus there exists a W-tile north of t and an N-tile west of t as drawn in Figure 2.9. By the definition of RC, A is produced. For all assembly sequences to A there exists a first tile u outside the gray region. u is either added as an N-tile east of t or a W-tile south of t violating our assumption that A is RC. Contradiction. ■

Given a tile system $\mathbf{T} = \langle T, \{S\}, g, 2\rangle$ and a full RC assembly A, we say that an N-tile $t = (\sigma_N, \sigma_E, \sigma_S, \sigma_W)$ is **T-determined** if $[t' = (\sigma'_N, \sigma'_E, \sigma_S, \sigma'_W) \in T \implies t' = t]$. That is, there is only one tile in T with the same south binding domain as t. We say that an NE-tile $[t = (\sigma_N, \sigma_E, \sigma_S, \sigma_W)$ is **T-determined** if $t' = (\sigma'_N, \sigma'_E, \sigma_S, \sigma_W) \in T \implies t' = t]$; and similarly for the other tile classes. We say that A is **T-deterministic** if all its tiles are **T**-determined. We omit **T** when it is clear from context.

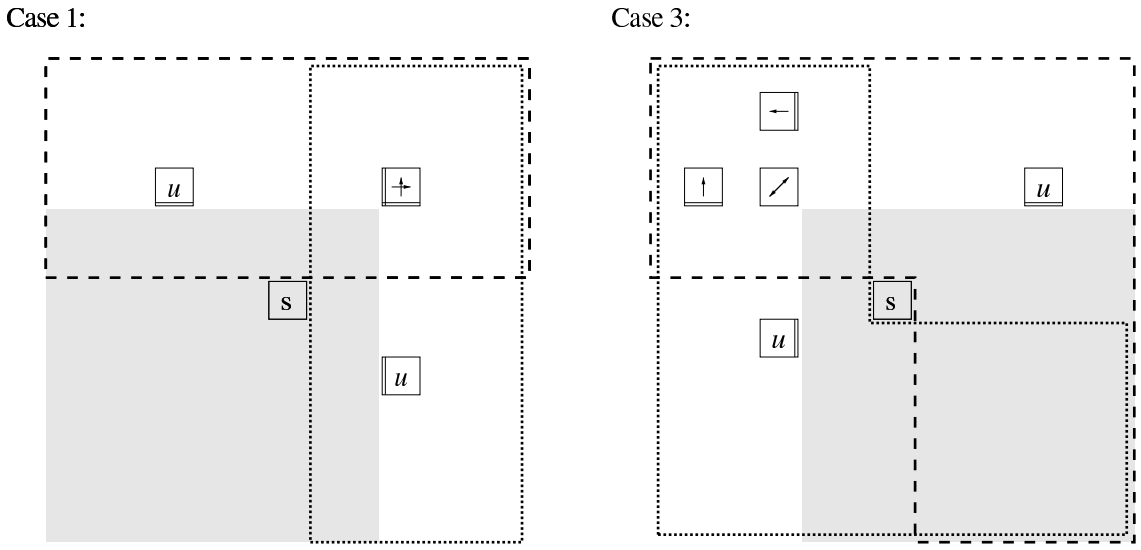


Figure 2.9: Every tile in an RC assembly is exclusively one of: the seed tile, an N-, NE-, E-, SE-, S-, SW-, W-, or NW- tile. Two cases from the proof. Case 1: N-tiles occur within the dashed line, E-tiles within the dotted line. A putative "N- and E-tile" occurs in the intersection of these regions and bears both north and east arrows. The first tile u to be added outside of the gray region must be an N- or E-tile, violating the RC property. Case 3: NE-tiles occur inside the dotted line, SW-tiles within the dashed line. A putative "NE- and SW-tile", occurs within the intersection of these regions and bears both northeast and southwest arrows; also it appears east of an N-tile and south of a W-tile. The first tile u to be added outside of the gray region must be an N- or W-tile, violating the RC property.

Borrowing from the study of polyominos [BM92] and lattice animals, we say that an assembly A is **convex** iff ($[A(x_1, y) \neq \textit{empty}$, $A(x_3, y) \neq \textit{empty}$ and $x_1 \leq x_2 \leq x_3]$ implies $A(x_2, y) \neq \textit{empty}$) and ($[A(x, y_1) \neq \textit{empty}$, $A(x, y_3) \neq \textit{empty}$ and $y_1 \leq y_2 \leq y_3]$ implies $A(x, y_2) \neq \textit{empty}$). That is, every row and every column in A contains just one contiguous block of tiles. Note that A may not be geometrically convex.

In the course of proving Lemma 12 below, we will see that, during the growth of a deterministic RC assembly, a tile of a given class is added by making bonds on the indicated sides (an N-tile makes a bond on its south side, an NE-tile make bonds on its

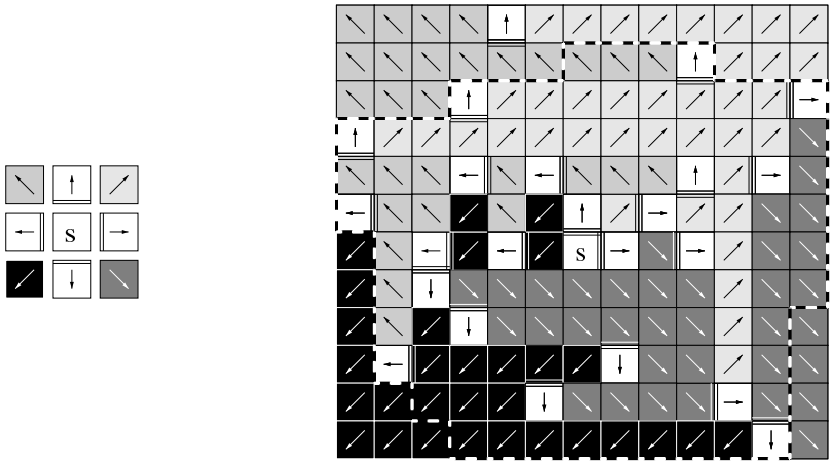


Figure 2.10: An RC labeling of a square assembly with N-, NE-, E-, SE-, S-, SW-, W-, and NW-tiles indicated by arrows. Tile classes are shown at left arranged according to their compass directions. "S" is the seed tile. A convex RC subassembly is indicated by the dotted line. Given that an RC assembly A is also $\mathbf{T}$-deterministic and in $Term(\mathbf{T})$, Theorem 9 guarantees that it will be uniquely produced by the tile system $\mathbf{T}$. Then the RC labeling can be thought of as a "map" for the partial order $\langle Prod(\mathbf{T}), \leq \rangle$: (i) $Prod(\mathbf{T})$ will contain exactly the convex RC subassemblies that can be drawn using the RC labeling as a reference and (ii) if B and C are convex RC subassemblies drawn using the RC labeling, $B \subseteq C$ will imply $B \leq C$.

south and west sides, etc.) More importantly, however, we are now ready to prove the main theorem of this section:

Theorem 9 $\forall\langle \mathbf{T}, A\rangle$ $\mathbf{T} = \langle T, \{S\}, g, 2\rangle$ *is a discrete tile system,* g *is non-negative, and* $A \in Term(\mathbf{T})$ *is a deterministic RC assembly:* $\mathbf{T}$ *uniquely produces* A*.*

This theorem follows trivially from:

Lemma 12 $\forall\langle \mathbf{T}, A\rangle$ $\mathbf{T} = \langle T, \{S\}, g, 2\rangle$ *is a discrete tile system,* g *is non-negative, and* $A \in Term(\mathbf{T})$ *is a deterministic RC assembly:* $C \in Prod(\mathbf{T})$ *implies that* C *is a convex deterministic RC assembly and that* $C \leq A$*.*

PROOF. The proof is by induction on the number of tiles in of C, denoted by $|C|$. The base case is trivial; $\mathbf{T}$ being singly seeded implies it is the seed assembly S. Suppose the lemma is true for all C with $|C| \leq n$, and $(S = C_1) \rightarrow_{\mathbf{T}} C_2 \rightarrow_{\mathbf{T}} \cdots \rightarrow_{\mathbf{T}} C_n \rightarrow_{\mathbf{T}} C_{n+1}$, where $C_n + \square_t^{(x,y)} = C_{n+1}$. Clearly, $|C_n| = n$, so C_n is a convex deterministic RC assembly and $C_n \leq A$. We will show that the same holds for C_{n+1} by considering the bonds that the new tile $t = C_{n+1}(x, y)$ makes with tiles from C_n.

Case 1: The new tile makes at least one strong bond; without loss of generality, assume a strong bond is made on the tile's south side.

To prove C_{n+1} satisfies the same properties as C_n we find it useful to prove that $y > 0$ and $C_{n+1}(x, y)$ is an N-tile. Say $y \leq 0$. Because $C_n \leq A$, the tile south of $C_{n+1}(x, y)$, $C_n(x, y-1)$, occurs in A as $A(x, y-1)$. Because $C_n(x, y-1)$ *can* bind $C_{n+1}(x, y)$ by a strong bond on the north, and A is a terminal full assembly, $A(x, y-1)$ *must* bind some tile via a strong bond on the north. Hence $A(x, y-1)$ is an S-tile. By the RC property C_n has an S-tile in row $y-1$, which cannot be $C_n(x, y-1)$ since $C_n(x, y)$ is empty. Hence

A must have two S-tiles in row $y-1$, violating the RC property of A. Thus $y > 0$ and $C_{n+1}(x,y)$ is an N-tile.

We now show $C_{n+1} \leq A$ and is deterministic. As above, $C_n(x,y-1) = A_n(x,y-1)$. Since the new tile $C_{n+1}(x,y)$ *can* be added to C_n north of the tile $C_n(x,y-1)$ via a strong bond, and A is a full terminal assembly, some non-empty tile $A(x,y)$ *must* bind north of $A(x,y-1)$ via a strong bond with the same binding domain. Thus, $A(x,y)$ is an N-tile and has the same binding domain as $C_{n+1}(x,y)$. With the assumption that A is deterministic (and hence $A(x,y)$ is determined), the last statement implies $A(x,y) = C_{n+1}(x,y)$. This, with the assumption $C_n \leq A$, implies $C_{n+1} \subseteq A$. C_{n+1} and A are both in $Prod(\mathbf{T})$ so by Lemma 2, $C_{n+1} \leq A$. Further, since $A(x,y)$ is determined, $C_{n+1}(x,y)$ is determined. This, with the assumption that C_n is deterministic, implies C_{n+1} is deterministic.

To show that C_{n+1} is convex, we first show that t is the only tile in row y. Assume that in C_{n+1} there exists a tile in row y other than $C_{n+1}(x,y)$. Such a tile must also be present in C_n and because C_n is RC and $y > 0$, there must be an N-tile $C_n(j,y)$ such that $j \neq x$ (because $C_n(x,y) = \textit{empty}$). $C_n(j,y)$ must also appear in A as an N-tile; since $A(x,y)$ is an N-tile this violates the RC property of A. Hence $C_{n+1}(x,y)$ is the only tile in row y of C_{n+1}.

Now assume C_{n+1} is not convex. Since C_n is convex, either row y or column x must no longer be contiguous because of $C_{n+1}(x,y)$. But it cannot be row y since $C_{n+1}(x,y)$ is the only tile in row y. It cannot be column x since $C_{n+1}(x,y)$ is bonded directly north of $C_{n+1}(x,y-1)$, the northmost tile of a contiguous column. Hence C_{n+1} is convex.

Now we show C_{n+1} is RC. We know already that t is the only tile in its row and is an N-tile; thus t's row is consistent with C_{n+1} being RC. And t makes no bonds to its left

or right so the number of E- and W- tiles in the column below and the column below's neighboring columns are the same as in C_n. The rows and columns of C_n already had single N-,E-,S-, or W-tiles as is appropriate with C_n being RC. Hence C_{n+1} is RC.

Case 2: The new tile makes no strong bonds; therefore, since $\mathcal{T} = 2$, it must make at least two weak bonds. Because C_n is convex, it cannot make bonds on opposite sides of the new tile; therefore, it must make exactly two weak bonds on adjacent sides of the new tile. Without loss of generality, assume they are on the south and west sides of the tile. Because C_n is convex, and $C_{n+1}(x, y)$ extends already existing contiguous blocks of tiles in row y and column x, C_{n+1} is convex. Because C_n is RC, row y and column x are non-empty, and no strong bonds are formed, C_{n+1} is RC.

To show C_{n+1} is deterministic, we first identify C_{n+1} as an NE-tile. Further, because C_n is convex and $C_n(x, y) = \textit{empty}$, there can be no tile in C_n north or east of (x, y). That is, there can be no tile u in C_n at position (x', y) with $x' \geq x$ or (x, y') with $y' \geq y$. Further, there can be no tile anywhere "northeast" of (x, y). That is, there can be no tile v at a position (x', y') with $x' \geq x$ and $y' \geq y$ because v must connect to C_n via a path containing a tile u. Since there is a tile at $(0, 0)$, this implies either $x > 0$ or $y > 0$.

Say $x > 0$. Since there can be no tile north of (x, y) there can be no E-tile north of $C_{n+1}(x, y)$. Since C_{n+1} is RC there must be an N-tile in row y and it is west of $C_{n+1}(x, y)$ since there can be no tile to its east. Thus $C_{n+1}(x, y)$ is an NE-tile. Similarly for $y > 0$.

Since the new tile $C_{n+1}(x, y)$ *can* be added to C_n at (x, y) via a weak bonds on the south and west, and A is a full terminal assembly, some non-empty tile $A(x, y)$ *must* bind via weak bonds on the south and west with the same binding domains. Since A is RC and it is assembled from C_n which is RC, the N- or E-tile which makes $C_{n+1}(x, y)$ an NE-tile

is present in A, and no new N- or E-tiles in row y or column x not present in C_n can be present in A. Thus, $A(x, y)$ is an NE-tile and has the same binding domain as the NE-tile $C_{n+1}(x, y)$. Now C_{n+1} is deterministic and $C_{n+1} \leq A$ as in Case 1.

By induction, all assemblies produced by $\mathbf{T}$ are convex, deterministic, RC and assemble to A. ■

And, deciding whether a particular assembly is deterministic RC can be done simply and quickly:

Theorem 10 *$\forall \langle \mathbf{T}, A \rangle$, $\mathbf{T} = \langle T, \{S\}, g, \mathcal{T} \rangle$ is a discrete singly-seeded tile system, A is an assembly: "A is a deterministic RC" can be decided in time $O(|T| + |A|)$.*

PROOF. We first decide whether A is RC. This requires verifying that there is exactly one N-tile in each row north of the seed tile, and so on for the S-, E-, and W-tiles. First we find the location of the seed tile $S(0, 0)$ in A and make sure that it occurs only once. Next we make a single scan through the assembly marking N-, S-, E-, and W-tiles, as defined, keeping a count of how many tiles of each type occur in each row and column (using up to $4|A|$ counters). After the scan we verify that the appropriate counters are exactly one. Thus deciding whether A is RC takes $O(|A|)$ steps.

We next decide whether A is deterministic. This first requires classifying the remaining tiles as NE-, NW-, SE-, or SW-tiles. A tile is an NE-tile if either (1) it is directly east of an N-tile and not directly south of a E-tile, or (2) it is directly north of an E-tile and not directly west of an N-tile. NW-, SE-, and SW-tiles require similar properties. Examining, for each tile in A, all the tiles in its row and column could incur $O(|A|^2)$ steps; instead we use a preprocessing step. We thus pass east to west and west to east through each row,

north to south and south to north through each column, in $O(|A|)$ steps, marking tiles as "east of an N-tile", "south of an E-tile", etc. Another single pass through the assembly allows us to mark the NE-, NW-, SE-, and SW-tiles in $O(|A|)$ steps.

Each non-seed tile in T may occur in 8 different contexts — as an N-tile, NE-tile, etc. Whether a tile is determined in a given context depends only on the tile set. Thus, by inspecting the tile set, we may build a 2D Boolean array indicating whether each tile-context combination would be determined if it appeared in an assembly. Then we may make a single scan through the assembly making sure that each tile only appears in a context that is determined. Preprocessing the tile set to build the list of determined tile-contexts can be done in $O(|T|)$ steps[36] and scanning the assembly takes $O(|A|)$ steps. Thus, deciding whether an assembly is deterministic RC takes $O(|T| + |A|)$ steps. ■

Our first design properties for unique production, which resulted in uniquely addressed assemblies, also guaranteed that the assemblies were produced and terminal (Theorem 8). This is not true for deterministic-RC; to see this, consider the two RC labelings in Figure 2.11[37]. Thus, including checks to see that A is produced and terminal, verifying that

[36]Building this list in $O(|T|)$ rather than $O(|T|^2)$ steps requires keeping a 1D array for strength-1 interactions and a 2D array for strength-2 interactions indicating whether a particular label or pair of labels has already been used in a given context and if so, by which tile.

[37]Given a particular RC labeling of interest, it is easy to determine whether there exists an assembly with that RC labeling which can be produced by *some* $\mathcal{T} = 2$ tile system: Inspect the RC labeling and construct a tile set with a tile for all inequivalent non-seed tiles where two tiles are equivalent iff their direction and bond interactions strengths match; add the seed tile to the tile set. Give all strength-1 edges the same label and all strength-2 edges a different label. Then run the algorithm from Theorem 2 using the RC labeling as a reference. Then an interesting question becomes, given an RC labeling consistent with *some* produced assembly for *some* $\mathcal{T} = 2$ tile system: "What is the minimum number of distinct tiles that are required in the construction of a *uniquely produced* assembly having the given RC labeling?" It is obvious that 10 tiles may be used to uniquely produce an assembly with the labeling in Figure 2.11a and that no fewer tiles can be used. It is less obvious how many tiles are required to uniquely produce an assembly with the RC labeling in Figure 2.10. This question seems very difficult — given an RC labeling like that in Figure 2.16 one has to find a tile system that does at least as well as our counter construction; even worse, given an RC labeling like that in Figure 2.19 it appears that one may have to find the most compact Turing machine construction consistent with the labeling.

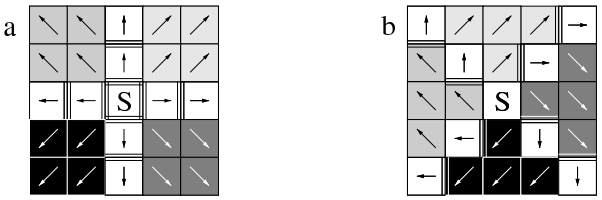


Figure 2.11: Deterministic-RC does not imply that an assembly is produced. (a) If deterministic, an assembly with this RC labeling would be produced by an appropriate tile system. (b) A similar RC labeling for which *no assembly*, deterministic or not, can be produced by a $\mathcal{T} = 2$ tile system.

an assembly designed to be deterministic-RC is uniquely produced takes only $O(|T||A|)$ steps.

We remark that an RC assembly is, by definition, uniquely seeded and that Lemma 12 requires that a tile system be singly seeded. Thus, by Theorem 1, a tile system which uniquely produces a deterministic-RC assembly A will have the nice property that $Prod(\mathbf{T})$ is isomorphic to $\widetilde{Prod(\mathbf{T})}$. Our claim that it is easy to design tile systems to produce deterministic RC assemblies is shown by example; in the next section we examine the program-size complexity of squares by demonstrating deterministic-RC constructions.

2.5 Program-size Complexity for Squares

Since we will be measuring program-size complexity using asymptotic notation, we begin by reviewing a few definitions. All functions will be from $\mathbb{N} \to \mathbb{N}$. A function $f(n)$ is **non-decreasing** iff $\forall n,\ f(n) \leq f(n+1)$. A function $f(n)$ is **unbounded** iff $\forall c,\ \exists n$ s.t. $f(n) \geq c$. We say $f(n) = O(g(n))$ iff $\exists c, n_0$ s.t. $\forall n > n_0,\ f(n) \leq cg(n)$. We say $f(n) = \Omega(g(n))$ iff $\exists c, n_0$ s.t. $\forall n > n_0,\ f(n) \geq cg(n)$. We assert proposition $P(n)$ **infinitely often** iff $\forall n_0 > 0, \exists n > n_0$ s.t. $P(n)$. Define $O_{i.o.}$ ("big-O infinitely often") such that $f(n) =$

$O_{i.o.}(g(n))$ iff $\exists c$ s.t. $f(n) \leq cg(n)$ infinitely often. We assert proposition $P(n)$ **for almost all** n iff $lim_{n_0 \to \infty} \frac{|\{1 \leq n \leq n_0 \text{ s.t. } P(n)\}|}{n_0} = 1$. Define $\Omega_{a.a.}$ ("big-Ω almost always") such that $f(n) = \Omega_{a.a.}(g(n))$ iff $\exists c$ s.t. $f(n) \geq cg(n)$ for almost all n.

We now formally describe the program-size complexity of an $N \times N$ square. An assembly A is an $N \times N$ **square** if there exists a site (x_0, y_0) such that $A(x, y) \neq empty$ iff $x_0 \leq x < x_0 + N$ and $y_0 \leq y < y_0 + N$. Thus the identity of tiles in a square is arbitrary. Square A is a **full square** if A is a full assembly; that is, neighboring tiles must make a bond with non-zero strength. We are interested in the set of squares which can be self-assembled by tile systems with temperature $\mathcal{T}$:

$$Sq^{\mathcal{T}} = \{(N, n) \in \mathbb{N} \times \mathbb{N} \text{ s.t. there exists a discrete tile system}$$
$$\mathbf{T} = \langle T, \square_s^{(0,0)}, g, \mathcal{T} \rangle, n = |T|, g \text{is diagonal}$$
$$\text{and } \mathbf{T} \text{ uniquely produces an } N \times N \text{ full square } \}.$$

We define the program size complexity $K_{Sq}^{\mathcal{T}}(N)$ of a square to be the minimum number of distinct non-empty tiles required to uniquely produce the square.

$$K_{Sq}^{\mathcal{T}}(N) = \min\{n \text{ s.t. } (N, n) \in Sq^{\mathcal{T}}\}$$

We begin by studying $K_{Sq}^{\mathcal{T}}(N)$ for $\mathcal{T} = 1$ and obtain the following theorem:

Theorem 11 $K_{Sq}^{1}(N) = N^2$.

PROOF. To show $K_{Sq}^{1}(N) \leq N^2$, we construct N^2 tiles, one for each position in the square, with a unique strength-1 binding domain for each neighboring pair of tiles as in

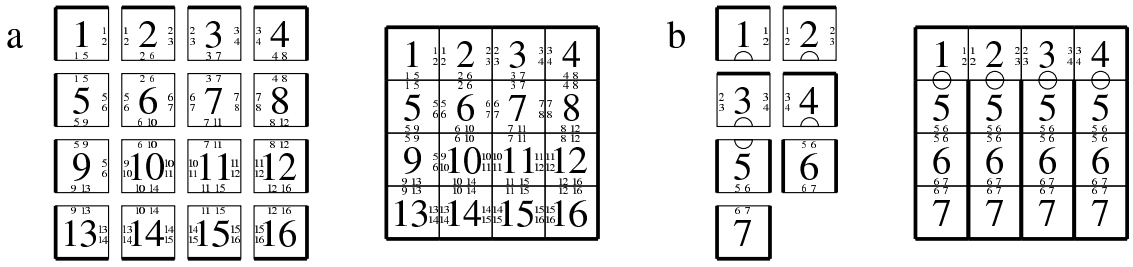


Figure 2.12: Formation of squares at $\mathcal{T} = 1$. For both constructions, let s be the tile labeled 1. (a) $N^2 = 16$ tiles with unique side labels uniquely produce a terminal 4×4 full square at $\mathcal{T} = 1$. (b) $2N - 1 = 7$ tiles uniquely produce a 4×4 square (but this is not a full square since thick sides have strength 0). Except for the sides labeled with a circle, each interacting pair of tiles share a unique side label. This comb-like construction is conjectured to be minimal for $N \times N$ squares assembled at $\mathcal{T} = 1$.

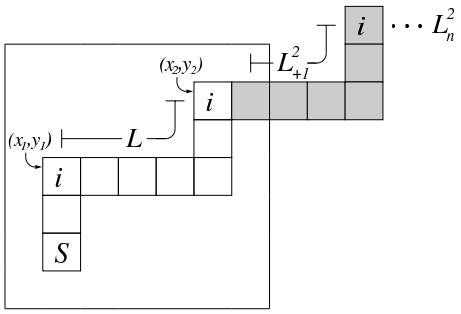


Figure 2.13: No $\mathcal{T} = 1$ tile system with fewer than N^2 tiles can uniquely produce an $N \times N$ square. A full $N \times N$ square with fewer than N^2 tiles must have some tile i present at two sites. Consider the assembly R (the white tiles) which includes an assembly L (bounded by the tiles i), the seed tile s, and a tile that connects the seed tile to L. R can be extended indefinitely with the addition of translated segments of L (*e.g.* L_{+1}^2 shown in gray).

Figure 2.12a. By Theorem 8 the square is uniquely produced. To show $K^1_{Sq}(N) \geq N^2$ we use a pumping argument. Suppose a tile set T with $|T| < N^2$ produces an $N \times N$ full square A (Figure 2.13). Then some tile i is present at two sites in A, say (x_1, y_1) and (x_2, y_2). Let L be the "L"-shaped (or possibly linear) assembly consisting only of the tiles at $(x_1, y_1) \ldots (x_2, y_1) \ldots (x_2, y_2)$; let L^1 be the assembly such that $L^1 + (x_2, y_2) = L$; let L^2 be the assembly such that $(x_1, y_1) + L^2 = L$; let $L^k_n(x, y) = L^k(x + n*(x_2 - x_1), y + n*(y_2 - y_1))$ be a translated version of L^k for $k = 1, 2$; and let R consist of L, s, and the fewest tiles in A required to connect s to L. Because R is contained in A and A is a full square, all neighboring pairs of tiles make a (non-zero strength) bond, and therefore $s \rightarrow^*_{\mathbf{T}} R$. At least one of $\{L^1_{-1}, L^1_{+1}, L^2_{-1}, L^2_{+1}\}$, say L^p_q, can be added to R, resulting in a larger assembly also produced by $\mathbf{T}$. This can be continued indefinitely: if $q = +1$ then for all n, $R + \sum_{i=+1}^{n} L^p_i$ is in $Prod(\mathbf{T})$; if $q = -1$ then for all n, $R + \sum_{i=-n}^{-1} L^p_i$ is in $Prod(\mathbf{T})$. This contradicts the assumption that $\mathbf{T}$ is halting and terminates in $N \times N$ full squares. ■

Thus, any uniquely produced full square at $\mathcal{T} = 1$ must be uniquely addressed, and any tile system that produces it cannot be tile-efficient[38]. At $\mathcal{T} = 2$ the situation is markedly different:

Theorem 12 $K^2_{Sq}(N) = O(N)$.

[38]Figure 2.12b shows that a uniquely produced, *non-full* square *can* be made tile-efficiently at $\mathcal{T} = 1$ using $2N - 1$ tiles. One may ask, "Why not ask about program-size complexity for non-full squares? Is the difference between complexity for $\mathcal{T} = 1$ and $\mathcal{T} = 2$ an artificial consequence of choosing to study *full* squares?" We do not believe that defining program-size complexity for non-full squares would change the overall picture much. First, we believe, but have not proven, that the non-full square complexity for $\mathcal{T} = 1$ is exactly $2N - 1$ tiles — perhaps practical but not a great improvement over N^2 tiles. Second, we know that the non-full square complexity for $\mathcal{T} = 2$ cannot be better than the Kolmogorov bound $\Omega(\frac{\log N}{\log \log N})$ already achieved in [ACGH01]. Also, full squares are interesting to us because we believe that, implemented in a physical system, full squares will be more mechanically stable than non-full squares.

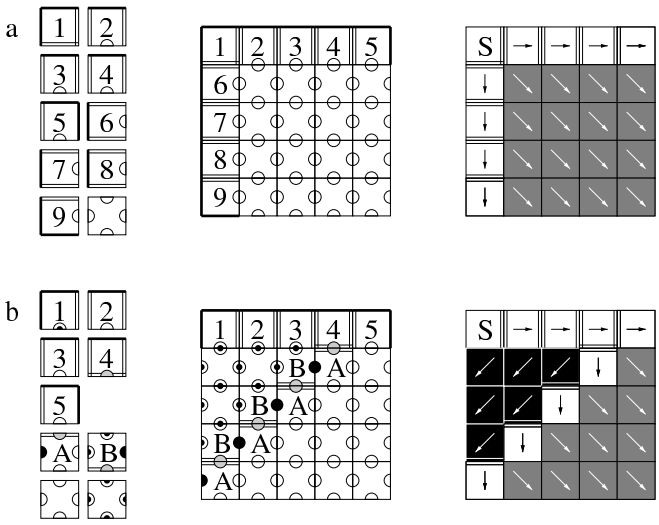


Figure 2.14: Formation of full squares at $\mathcal{T} = 2$. At left, tile sets; in the middle, square assemblies; at right, labeling to indicate the assembly is RC as in Figure 2.10. Again, let the tile labeled 1 be the seed tile. (a) $2N = 10$ tiles uniquely produce 5×5 full square. Except for the sides labeled with a circle, each interacting pair of tiles share a unique side label (but we do not label them explicitly as in Figure 2.12.) (b) $N+4 = 9$ tiles are used.

PROOF. Figure 2.14 shows two constructions for an $N \times N$ full square using $2N$ (Figure 2.14a) and $N+3$ (Figure 2.14b) tiles respectively. In Figure 2.14a, self-assembly starts from the seed tile 1 and proceeds by single strength-2 interactions between numbered tiles to create an L-shaped border for the square. As the border grows, pairs of cooperative strength-1 interactions allow the blank tiles to fill in and complete the square.

In Figure 2.14b, only one edge is created by numbered tiles via strength-2 interactions. Then the **A** and **B** tiles build a diagonal, entering a new column by their strength-2 sides, thus allowing the rest of the column to be filled with blanks.

For the *particular instances* of the constructions shown in Figure 2.14, say $N = 5$ as shown, it is easy to verify that the squares are uniquely produced by running the algorithms from Theorems 2, 3 and 10 to verify that the assemblies shown are produced, terminal,

and deterministic RC. RC labelings for both sample constructions are given at right in Figure 2.14.

That our constructed assemblies are produce, terminal and deterministic RC *for all N* (and hence uniquely produced) requires performing induction on our construction. This is simple but requires representing the construction more abstractly than in Figure 2.14; thus we omit the proof and leave it as an easy exercise for the reader. ■

An interesting aside: for construction such as this, we like to think of the infinite family of tile systems defined by the construction as a self-assembly program (in contrast to those times when we think of a single tile system as a self-assembly program). We note that the tile sets for such constructions can be divided conceptually into two parts: (i) a constant part, which we refer to as *rule tiles* (the blank tiles and diagonal tiles in Figure 2.14) and (ii) a variable part, which we refer to as *border tiles* or *input tiles* (the numbered tiles in Figure 2.14). This conceptual division corresponds to the division between *program* and *data* for computer programs.

The constructions in Figure 2.14 achieve tile-efficiency by uniquely addressing a partial border for a square and using cooperative binding to fill in the rest. In forming the border, the tiles can be said to count in unary, up to the width of the square. This suggests that the construction in Figure 2.14b can be combined with a fixed-width version of the binary counter of Figure 2.4 to obtain a set of tiles that produce the $N \times N$ full square by counting in binary instead of by counting in unary. By such a construction we show:

Theorem 13 $K^2_{Sq}(N) = O(\log N)$.

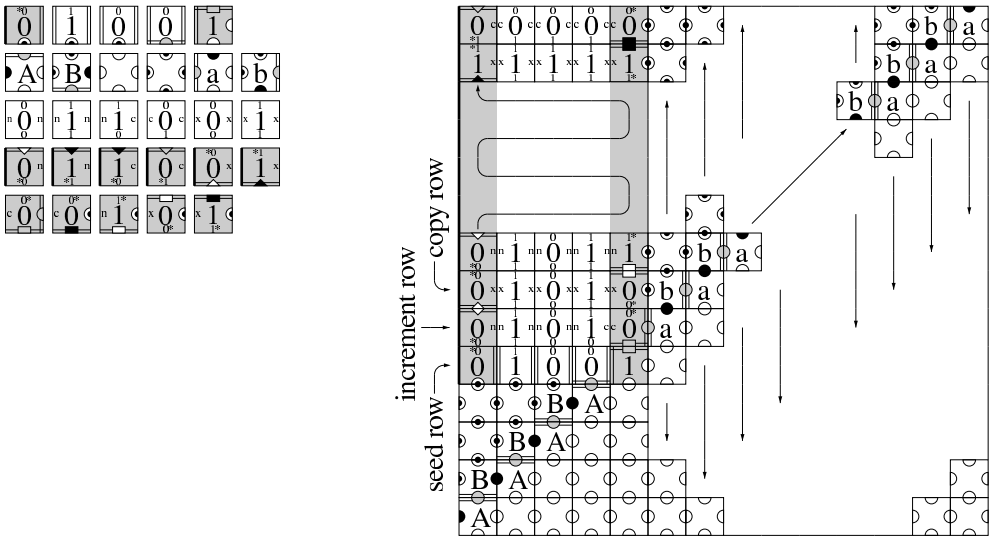


Figure 2.15: Formation of $N \times N$ square using $O(\log N)$ tiles. Construction starts with an $n-1 \times n-1$ square as in Figure 2.14b. Here $N = 52, n = 6$ and 28 tiles are used. The construction illustrates the case for even $N-n$; the first row above the seed row is a copy row for odd $N-n$.

PROOF. Figure 2.15 constructs an $N \times N$ full square using $n+22$ tiles, where $n = \lceil \log N \rceil$. $n+2$ tiles, including the seed tile, produce an $(n-1) \times (n-1)$ square as in the previous construct. Additionally, the $n-1$ tiles in the seed row have upper sides encoding the bits of the integer $c = 1 + 2^{n-1} - (N-n)/2$, the initial value of the counter. We must use a fixed-width version of the counter tiles of Figure 2.4; this requires a special set of tiles for the leftmost and rightmost columns of bits. The counter counts from c to 2^{n-1}, using two rows for each integer. In order to detect when the counter has finished, we use alternating rows to increment the counter from right to left, then to copy the the bits from left to right *unless* the leftmost bit just rolled over from 1 to 0. In the latter case, the tile presents a strength-2 side with a label not found on any other tiles, thus halting the counter. (The strength-2 side will be used in our next construction; here, any strength would suffice.)

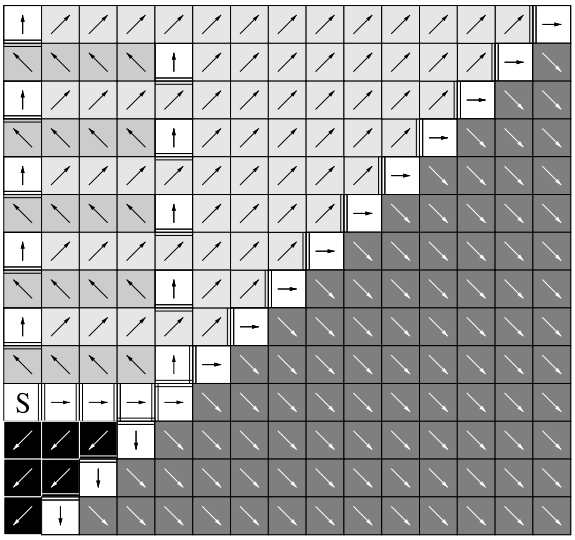


Figure 2.16: RC labeling of one instance of the $O(\log N)$ construction. Note two features of interest. First, all N-tiles appear in the counter — its growth limits growth of the square to the north. Second, the same tile can appear in more than one directional context in a deterministic-RC assembly: the blank tile with a black dot on its edge appears as both SW and NE tiles in this construction. In this construction it is obvious that the order of constructing the square is not unique. Part or all of the counter can be built, for example, before the portion of the square below the seed row.

There is a special tile for the rightmost bit in the first increment row above the seed row. This tile contains a strength-2 side to initiate the **a-b** diagonal, thus filling in the rest of the square. Overall, the counter requires 18 tiles; the seed row requires $n-1$ tiles; the two diagonals require 4 tiles; and there are two blank tiles.

Again, the unique production of any particular instance of this construction can be easily verified by running algorithms (as in Theorem 12. An RC labeling for a sample construction is given in Figure 2.16. Again we omit the induction showing that the construction is uniquely produced for all N. ■

The above construction works for all N. For an infinite set of special N we can do much better: by recursively iterating the above construction one can produce $N \times N$ squares with

$$N \geq \underbrace{2^{2^{2^{\cdots^{2}}}}}_{n \text{ times}} \stackrel{def}{=} 2 * *n$$

using only $O(n)$ tiles. Define $\log^* N$ as the least n such that $2 * *n \geq N$.

Theorem 14 $K^2_{Sq}(N) = O_{i.o.}(\log^* N)$.

PROOF. Our proof is by induction. Let S^n refer to a tile system containing fewer than $21n$ tiles (including the **a**, **b**, and *blank* tiles) that uniquely produces an $N_n \times N_n$ full square such that:

- $N_n > 2 * *n$.
- All binding domains on the west and south are of strength 1 or strength 0.
- All binding domains on the east have the strength-1 blank label.

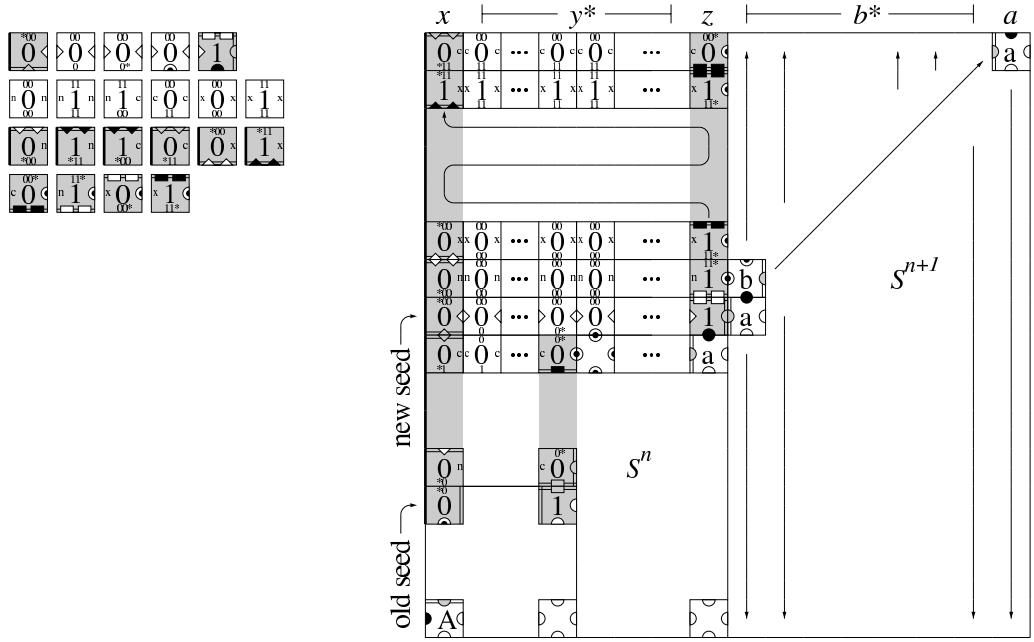


Figure 2.17: Formation of $N \times N$ square using $O_{i.o.}(\log^* N)$ tiles. Given a set of tiles S^n that produce an $N_n \times N_n$ full square that satisfies the recurrence, the addition of 21 new tiles results in S^{n+1} and produces an $N_{n+1} = (N_n + 2 \times 2^{N_n}) \times (N_n + 2 \times 2^{N_n})$ full square. New side labels (with doubled symbols) prevent counter tiles from S^n from incorporating in the S^{n+1} counter.

- The binding domains on the north conform to the pattern xy^*zb^*a where x is a strength-2 binding domain that occurs nowhere else, and y, z, b, and a are distinct strength-1 binding domains.

We show that S^n exists for all n. The base case $n = 1$ is trivial. The inductive step is illustrated in Figure 2.17. We create S^{n+1} by adding 21 tiles to S^n as follows. First, there are 5 tiles that, initiated by x, produce an initial string of 0's for a new fixed-width counter, and provide a strength-2 side for a new **a-b** diagonal. Then there are 16 tiles equivalent to the counter tiles in Theorem 3 but using new side labels; the counter counts to 2^{N_n}. The diagonal fills in the rest of the square, now with sides of length $N_{n+1} = N_n + 2 \times 2^{N_n} > 2^{N_n} > 2**(n+1)$. Therefore S^n exists for all n.

The existence of S^n for all n, implies that for infinitely many N_n,

$$K^2_{Sq}(N_n) \geq 21n \geq 21 \log^* N_n.$$

Again, the unique production of any particular instance of this construction can be easily verified by running algorithms (as in Theorem 12. And again we omit the induction showing that the construction is uniquely produced for all N. ■

$\log^* N$ is an exceedingly slowly growing function; the above construction shows that very large squares can be assembled with a very small number of tiles. But we can do much better yet! By embedding the simulation of a Turing machine in the growth of a square we show that:

Theorem 15 $K^2_{Sq}(N) = O_{i.o.}(f(N))$ *for* $f(N)$ *any non-decreasing unbounded computable function.*

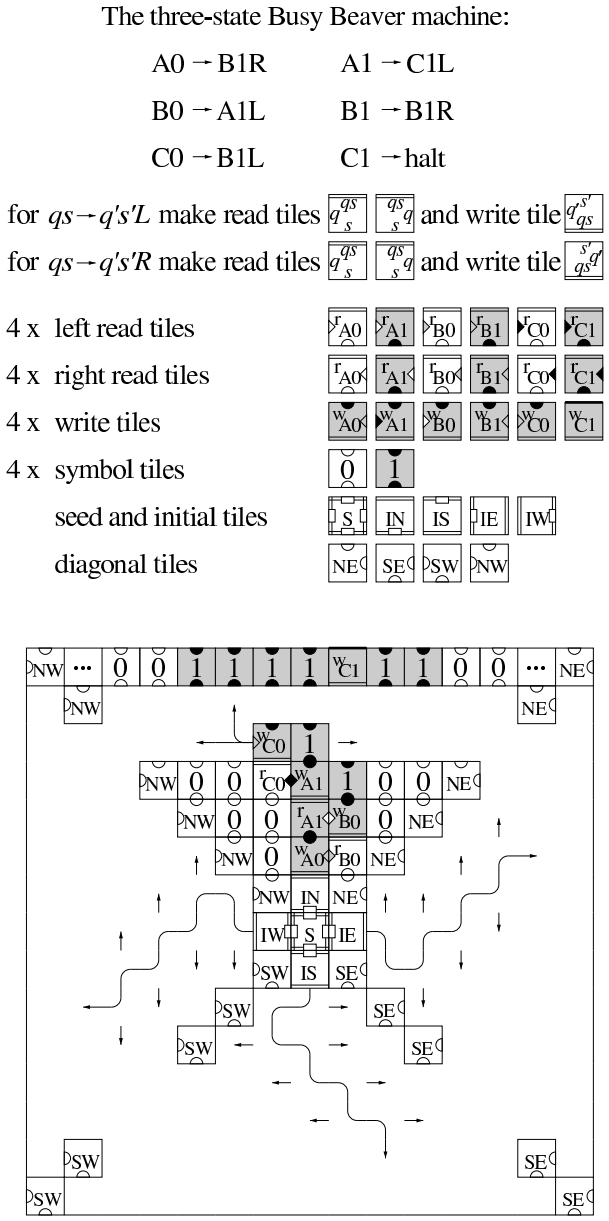


Figure 2.18: Formation of an $N \times N$ square by growing four identical simulations of a given Turing machine. The Busy Beaver machine simulated here has three states ($q_0 = A, q_1 = B, q_2 = C$) and two symbols ($s_0 = 0, s_1 = 1$). Note that R denotes right, L denotes left, and "4x" indicates that four variations of a tile are used, one for each compass direction.

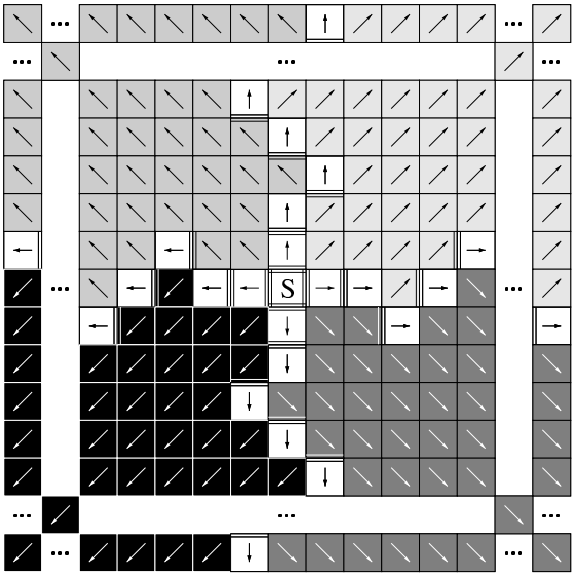


Figure 2.19: RC labeling for one instance of the Turing machine construction.

PROOF. Our proof relies on a self-assembly version of the Busy Beaver problem [Rad62]. Define:

$$B^{\mathcal{T}}_{Sq}(n) = \max\{N \text{ s.t. } (N, n) \in Sq^{\mathcal{T}}\}.$$

To show Theorem 15, we first show

$$B^{2}_{Sq}(n) = \Omega(F(n)) \text{ for any computable function } F(n). \tag{2.1}$$

Theorem 15 follows from (2.1) by contradiction: if false, then there exists a computable, non-decreasing, unbounded function $f(N)$ such that $\exists N_0$ s.t. $\forall N > N_0,\ K^{2}_{Sq}(N) \geq f(N)$. Let $F(n) = \max\{N \text{ s.t. } N = 0 \text{ or } f(N) \leq n\}$; this is a computable function. Note that $B^{2}_{Sq}(n) \geq F(n)$ requires that $\exists (N, n) \in Sq^2$ s.t. $N \geq F(n)$ and therefore $f(N) > n$ and

$K^2_{Sq}(N) \leq n$. For $N > N_0$ this contradicts $K^2_{Sq}(N) \geq f(N)$. Therefore, for all $n > f(N_0)$, $B^2_{Sq}(n) < F(n)$, contradicting (2.1) and establishing Theorem 15.

Recall that $B_t(m) = \Omega(F'(m))$ for any computable function $F'(m)$ where:

$$B_t(m) = \max\{t \text{ s.t. } m = qs \text{ and there exists a } q\text{-state, } s\text{-symbol Turing machine that halts on a blank tape in } t \text{ steps}\}$$

Let M be a q-state, s-symbol Turing machine that halts on a blank tape in $B_t(m)$ steps, where $m = qs$. We will construct a square of size $N = 2B_t(m) + 3$ using $n = 12qs + 4s + 9$ tiles by simulating M with tiles, similar to the construction of Robinson [Rob71]. Given any $n > 41$, we will use $s_n = 2$, $q_n = \lfloor \frac{n-17}{24} \rfloor$, and $m_n = q_n s_n$; our construction will need only $12q_n s_n + 4s_n + 9 < n$ tiles. Then $B^2_{Sq}(n) \geq 2B_t(m_n) + 3 = \Omega(F'(m_n))$. For any computable function $F(n)$, we can find another computable function $F'(m)$ s.t. $\forall n,\ F'(m_n) > F(n)$. Therefore, we arrive at (2.1).

We construct the square by growing four identical simulations of the Turing machine M, one from each side of a seed tile. Each simulation stays within one of the four regions bounded by the diagonals of the square; when M halts, the square is complete. We require 4 tiles to create the four "half-diagonals" defining these boundaries between simulations. For each simulation we require 1 "initial state" tile that matches the seed tile, s symbol tiles, qs write tiles, and $2qs$ read tiles, giving a total of $3qs + s + 1$ tiles per simulation. We describe these tiles for the TM simulation to the north of the seed tile. Recall that a tile is a 4-tuple $(\sigma_N, \sigma_E, \sigma_S, \sigma_W)$ representing the north, east, south, and west binding domains. Binding domain strengths are 1 unless noted. Each of the four simulations has

its own version of the side labels described, distinguished by superscripts (we omit the superscript ^{N} from the description of north-facing simulation below). The symbol tile for symbol s is $(\sigma_s, \sigma_e, \sigma_s, \sigma_e)$, where σ_s is a binding domain representing the symbol s and σ_e is a binding domain indicating that the TM head is not present. For each state-symbol pair (q, s), the left read tile $(\sigma_{q,s}, \sigma_e, \sigma_s, \sigma_q)$ and the right read tile $(\sigma_{q,s}, \sigma_q, \sigma_s, \sigma_e)$ represent the TM head in state q entering a tape cell (from the left or from the right) and reading the symbol s. The binding domains $\sigma_{q,s}$ have strength 2; this is necessary for the TM head to enter the next row of the simulation. The write tiles, representing the action the TM head takes, depend on the form of the state transition table entry. For each entry of the form $(q, s) \to (q', s', L)$ there is a write tile $(\sigma_{s'}, \sigma_e, \sigma_{q,s}, \sigma_{q'})$; for each entry of the form $(q, s) \to (q', s', R)$ there is a write tile $(\sigma_{s'}, \sigma_{q'}, \sigma_{q,s}, \sigma_e)$; for each entry of the form $(q, s) \to halt$ there is a write tile $(\sigma_{halt}, \sigma_e, \sigma_{q,s}, \sigma_e)$.

To start the Turing machine in state q_0 reading the blank symbol s_0, the initial tile for the northern simulation is $IN = (\sigma_{q_0,s_0}, \sigma_e, \sigma_S, \sigma_e)$, where σ_S is a strength-2 binding domain. The initial tiles for all four simulations bind to the seed tile $S = (\sigma_S^N, \sigma_S^E, \sigma_S^S, \sigma_S^W)$. The four diagonal tiles, $NW = (\sigma_{s_0}^N, \sigma_e^N, \sigma_e^W, \sigma_{s_0}^W)$, $NE = (\sigma_{s_0}^N, \sigma_{s_0}^E \sigma_e^E, \sigma_e^N,)$, $SE = (\sigma_e^E, \sigma_{s_0}^E, \sigma_{s_0}^S, \sigma_e^S)$, and $SW = (\sigma_e^W, \sigma_e^S, \sigma_{s_0}^S, \sigma_{s_0}^W)$ pad the tapes with extra cells containing the blank symbol s_0 and delimit the four simulations.

Given a particular assembly for a particular Turing machine that halts, the unique production of the assembly can easily be verified by running algorithms as in Theorem 12. Figure 2.19 gives an RC labeling of the Turing machine simulation in Figure 2.18. In general, without knowledge that an embedded Turing machine halts, we cannot know whether or not a square is uniquely produced. It *can* be easily shown that for the *general* Turing

machine construction, a square is uniquely produced iff the embedded Turing machine halts; again we omit the proof. ■

Theorem 14 gives the construction of infinite number of very large squares made from a very small number of tile types. Theorem 15 implies that, for an infinite set of N, the number of tiles required to assemble an $N \times N$ square can be made "arbitrarily small". How well can one do in general? Unfortunately by results in Kolmogorov complexity, extremely concise self-assembly programs are not common. The essential idea is that a small tile system that uniquely produces an $N \times N$ square can be used to construct a small Turing machine that writes "N" on its tape. If a large fraction of integers are produced by small tile systems, then a large fraction of integers can be compressed by Turing machines — but this is impossible. Using this argument we show:

Theorem 16 $K^2_{Sq}(N) = \Omega_{a.a.}(\frac{\log N}{\log \log N})$.

PROOF. The Kolmogorov complexity of an integer N with respect to a universal Turing machine U is

$$K_U(N) = \min |p| \text{ s.t. } U(p) = \#N$$

where $\#N$ is the binary string representing N. (See [LV97] for results on Kolmogorov complexity.) Recall that $K_U(N) < \lceil \log N \rceil - \Delta$ for at most $\frac{1}{2}^{\Delta}$ of all N, by the pigeonhole principle. Therefore, for any $\epsilon > 0$, $(1 - \epsilon) \log N < K_U(N)$ for almost all N.

There exists a Turing machine $Sq2$ (with program p_{Sq2}) that, given a binary description of a $\mathcal{T} = 2$ tile system, simulates self-assembly starting from an arbitrary seed tile, making an arbitrary choice when multiple tile additions are possible, and returns the maximal dimension of the resulting assembly if self-assembly terminates. A discrete tile system

$\mathbf{T} = \langle T, \{S\}, g, 2\rangle$ with a non-negative diagonal g and $n = |T|$ tiles can be described by $d_{\mathbf{T}}$ containing $f(n) = 4n\lceil \log 3n \rceil$ bits; each of $4n$ sides may have a strength in $\{0, 1, 2\}$, and non-zero-strength labels may be defined by the first tile (in some arbitrary order) with the same label on the opposite side. For all N let $\mathbf{T}_N$ be a smallest tile system that uniquely produces an $N \times N$ full square; thus $K^2_{Sq}(N) = |\mathbf{T}_N|$. Then $p_N = p_{Sq2} d_{\mathbf{T}_N}$ will return $\#N$ when input to U. But p_N is at least as big as $K_U(N)$. Therefore, for almost all N,

$$\begin{aligned}(1-\epsilon)\log N < K_U(N) \leq & |p_{Sq2}| + |d_{\mathbf{T}_N}| \\ \leq & |p_{Sq2}| + f(K^2_{Sq}(N)) < C_1 + C_2 K^2_{Sq}(N)[C_3 + \log\log N]\end{aligned}$$

where we used $K^2_{Sq}(N) = O(\log N)$ from Theorem 13. The final result follows from simple algebra. ■

In other words, a Kolmogorov-random integer N cannot be compressed by the self-assembly model.

In the time since the paper [RW00] on which this chapter is based was originally written, Adleman *et al.* and others [ACGH01] have improved our bound on the complexity of squares to reach the lower bound dictated by Kolmogorov complexity. By using tiles to change the base representation of numbers, they have created a construction for which $K^2_{Sq}(N) = O(\frac{\log N}{\log\log N})$. This result is quite satisfying: in combination with Theorem 16 it makes the complexity of self-assembled squares, in a sense, asymptotically tight.

2.6 Variations on a Theme.

2.6.1 Disintegrating tiles.

A main conclusion of this chapter is that the program-size complexity of self-assembled objects (at $\mathcal{T} = 2$) looks remarkably similar to the usual program-size complexity with respect to Turing machines. This is hardly surprising, since self-assembly at $\mathcal{T} = 2$ can simulate Turing machines. However, Figure 2.20 makes K^2_{Sq} look perhaps more similar to K_U than it ought to: K^2_{Sq} is computable whereas K_U, of course, is not. K^2_{Sq} due to the monotonic nature of self-assembly: we can enumerate tile sets based on size, and, in turn, simulate each tile set until it either halts or exceeds an $N \times N$ region; the size of the first tile set to halt yielding an $N \times N$ square is K^2_{Sq}.

The difference between K^2_{Sq} and K_U comes from the monotonic nature of self-assembly: the growing object always gets bigger, so "temporary results" cannot be larger than the object itself. A simple device circumvents this difficulty: select a subset $D \subset T$; after self-assembly is complete, the tiles in D are destroyed in all assemblies in $Term(\mathbf{T})$, and the resulting assemblies are considered the output of the computation. A molecular implementation might make the tiles in D out of RNA, while the tiles in $T \setminus D$ are DNA; then, an RNase enzyme can be used to destroy all tiles in D.

Let $\hat{K}^2_{Sq}$ be the full square complexity for the model with disintegration. The ability of $\mathcal{T} = 2$ self-assembly to simulate Turing machines can now be used to make squares: given a Turing machine program p such that $U(p) = \#N$, we generate $C + |p|$ tiles (all in D) that simulate p and expand the result into a template of length N. Now a few final tiles (not in D) complete the square; only this square will survive disintegration. Thus

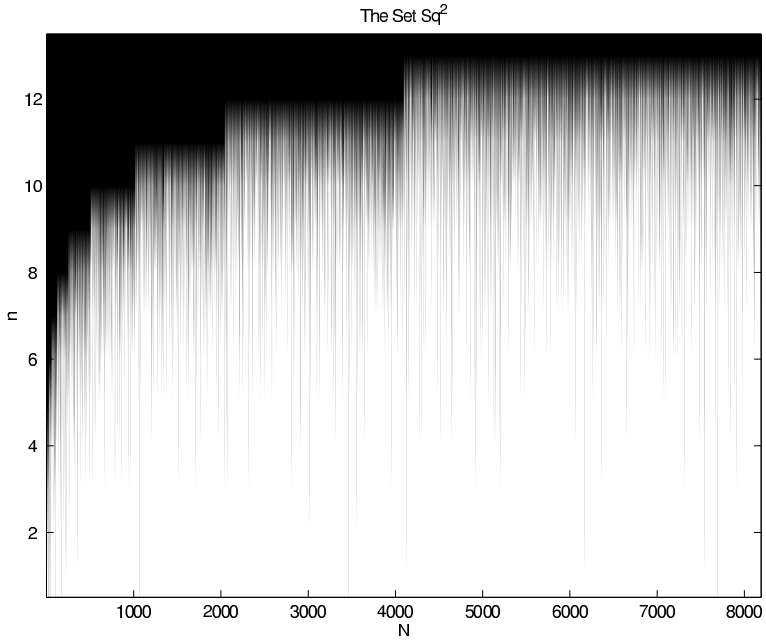


Figure 2.20: Artist's impression of the set Sq^2 of pairs (N, n) where n tiles produce an $N \times N$ full square(black). $K^2_{Sq}(N)$ is the lowest point in a column; the vertical lines are due to the fact that $(N, n) \in Sq^2 \implies (N, n+1) \in Sq^2$. $B^2_{Sq}(n)$ is the rightmost point in a row. From this chapter we know that for almost all N, $K^2_{Sq}(N)$ is bounded below by $C \log N / \log\log N$. We now know from [ACGH01] that $K^2_{Sq}(N)$ is bounded above by $C \log N / \log\log N$.

$\hat{K}^2_{Sq} < K_U(N) + C$; we already know $K_U(N) = O(\hat{K}^2_{Sq}(N) \log\log N)$. That is, the only difference is that $\hat{K}^2_{Sq}$ measures the number of tiles instead of the number of bits required to specify the tiles.

2.6.2 Tiles versus labels versus bits.

In this chapter we defined program-size complexity of an assembly as the minimum number of distinct types of tiles n required to uniquely produce the assembly. For self-assembly, the number of distinct tiles also coincides with what we defined as synthetic complexity — a measure intended to be a rough estimate of how practical a particular self-assembly scheme will be to implement. We phrased the motivation as the cost of buying n tiles but it could have equally well been the difficulty of synthesizing n different types of tiles themselves. Underlying this motivation is the assumption that the difficulty of synthesizing an n tile set increases linearly with the number of tiles; perhaps this is not so.

One requirement for our self-assembly programs to work correctly is that the binding domains (represented by labels) on the tiles correctly implement the strength function. In particular, for the strength functions that we study, it is important that a binding domain bind to itself and no other binding domain. But designing a set of such specific binding interactions seems difficult in practice. Consider the nth binding domain added to such a set — it must not only bind well to itself but *not bind* to $n-1$ binding domains already in the set. Thus it is easy to imagine that *designing* a set of such binding interactions becomes superlinearly more difficult in the number of binding domains.

We further imagine that *constructing* such a set physically *could be* similarly difficult. For diagonal strength functions we know that the information content of a set of n tiles

is $O(n \log n)$ bits rather than n bits because we must specify the labels on the tiles — we have to use this measure when rigorously comparing self-assembly to other models of computation. *It seems that whatever binding domains we use on the tiles, they must encode* $O(\log n)$ *bits of information.* If we assume that the size (in atoms) of the binding domains must increase linearly with the number of bits they encode, then each tile must incur $O(\log n)$ cost (microscopically in atoms, macroscopically in mass, and perhaps also in economic cost). Certainly, one natural way of creating specific binding interactions, the use of DNA sticky ends of length $O(\log n)$, would incur this cost. Thus we suspect that the number of *bits*, rather than the number of tiles or side labels, actually gives a better measure of the difficulty of obtaining a tile set. It is difficult to prove this suspicion because it is difficult to gauge the difficulty of using geometry rather than numbers and identities of atoms to encode bits. However, it motivates a more general question which we think is important: "Can we make the relationship between abstract complexity measures and the actual physical difficulty of constructing molecules precise and concrete?"

We note another way that informational cost be related to physical cost for tile construction. Consider the problem of modifying a tile-like molecule that has four identical binding domains A to create a molecule with exactly two binding domains B on opposite sides of the molecule. In practice it is difficult both (i) to modify the molecule so that exactly two As are changed to Bs and (ii) to specify that the Bs occur on opposite sides of the molecule. In reality all possible combinations of As and Bs are likely to appear on modified molecules and a purification (itself difficult because the molecules are so similar) must be performed. The physically difficult problem of *breaking symmetry* can be cast information theoretically — in making the molecule asymmetric we are decreasing the

amount of information (in bits) that we must specify to identify a particular side of the molecule.

2.6.3 Non-diagonal strength functions.

Pablo Moisset and Qi Cheng ([CdE01]) have explored non-diagonal strength functions and found that they can take optimal advantage of the additional information encoded in such functions — in doing so they find that the program size complexity of squares can be even smaller: $O(\sqrt{\log N})$. It remains to be seen how easily non-diagonal strength functions can be realized in the lab. One approach would be to assign each label σ_1 a set of binding domains, one for each non-zero entry $g(\sigma_1, \sigma_2)$ in the strength function g. A tile side labeled σ_1 would thus bear a *set* of binding domains. For currently used DNA motifs, however, this approach might make the connections between tiles too floppy to form ordered lattices.

2.6.4 Negative interaction strengths.

Many of our proofs rely on the fact that bond strengths are non-negative. In particular, this means that once a site (x, y) capable of binding a tile t is formed on an assembly, tile t can always bind at that site, so long as it remains open. If we allow negative interaction strengths, subsequent tile additions neighboring (x, y) may prevent t from binding. The questions of whether an assembly is produced, terminal, or uniquely produced remain decidable given negative interaction strengths but whether they remain decidable in polynomial time is not known.[39]

[39]With Matt Cook, we have found a construction for a 3D tile systems with negative bond strengths to which 3-SAT may be reduced — in fact the construction requires only two layers of tiles. Thus, in 3D the

Although we know little about the implications of negative bond strengths, it appears that they may have some advantages. For example, if we could give mismatched labels interaction strengths of -1, our ability to achieve perfect $\mathcal{T} = 2$ conditions for our $\mathcal{T} = 2$ constructions might become less important — the repulsive force between mismatched sides might help ensure that a tile that binds a corner site by two bonds is greatly preferred over a tile that binds the corner site by a single bond.

Negative interaction strengths were not included in the original formulation of the Tile Assembly Model because it seems that they are more difficult to implement in the real world. For example, we have no idea how to achieve negative interaction strengths for DNA molecules using normal DNA bases; many mismatched base pairs have weakly positive interactions. Molecular self-assembly based on electrostatic interactions might more easily yield negative interaction strengths[40]. For capillary force self-assembly, explored in Chapter 3, it is possible to create interactions between mismatched labels that should, theoretically, be of strength $-2/3$. The interactions are indeed repulsive, as demonstrated in Figure 3.3, but the interaction strength has not been measured experimentally.

The Tile Assembly Model with negative interaction strengths does seem like a good model for steric (geometric) effects in self-assembly; reciprocally, steric effects may be used to implement the strength -1 bonds we desire. For an example, see Figure 2.21 in which shape complementarity is used to create bond specificity. Our current DNA motifs cannot create such steric effects; single stranded DNA strands that mismatch will just fold out

question of whether an assembly is produced is $\mathcal{NP}$-complete and the question of whether an assembly is uniquely produced is co-$\mathcal{NP}$-complete.

[40]Negatively charged DNA backbones repel each other if positive counterions like Mg^{2+} are not present to screen their interactions. It may be possible to tune the ionic strength of a solution so that interactions between DNA strands are negative unless base pairing interactions overcome backbone repulsion.

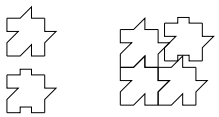


Figure 2.21: Negative bond strengths may model steric effects. A tile that matches a site on one side but not the other cannot bind — geometrical incompatibility at the mismatched side precludes binding at the matched side because the tile is rigid.

of the way. Further, we are not aware of a class of synthetic molecules that would allow us to put such schemes into practice. In principle, however, there is no difficulty since shape complementarity appears to play a large role in the formation of natural protein complexes. The rigid tiles used in capillary force self-assembly seem ideal for using shape complementarity; so far, however, attempts to create multiple pairs of specific interactions based on shapes have been unsuccessful (Chapter 3, p. 151).

2.6.5 Temperature 1 versus Temperature 2.

In introducing the Tile Assembly Model we discussed that tiles under $\mathcal{T} = 2$ conditions could exhibit *cooperative* binding. This meant that, where a single strength-1 interaction was too weak to hold a tile in a corner site, a *pair* of strength-1 interactions could "co-operate" to hold the tile in place. In a couple senses, cooperative binding allows a logical AND to be performed. For a tile to bind to a corner site presenting strength-1 binding domains A and B, the tile must have both binding domain A *and* binding domain B. This trick allows a function of two inputs to be computed, as for the counter tiles. Turned upside down, *no* tile can bind at an empty site until *both* binding domains A *and* B are present, which means that *both* tiles adjacent to the site must already be present. This trick allows two-dimensional growth to be delimited by borders, as in Figure 2.14. More

abstractly it allows the execution of self-assembly programs to be ordered, so that one part (or subroutine) of an assembly is guaranteed to be completed before another part of the assembly requires it. Used in these ways, cooperative binding is the single most important trick by which our $\mathcal{T} = 2$ constructions compute, and it is the single most important trick by which we arrive at an efficient program size complexity for squares.

If we wish to implement our $\mathcal{T} = 2$ constructions experimentally and take advantage of cooperative binding, we must carefully tune temperature, concentration and bond strengths to maintain $\mathcal{T} = 2$ — any time a tile binds by a single strength-1 bond (a "$\mathcal{T} = 1$ event") it is very probable that our carefully designed $\mathcal{T} = 2$ constructions will fail. Choosing correct conditions is a delicate balancing act [Win98a]: high concentrations favor $\mathcal{T} = 1$ events, but low concentrations decrease the speed of assembly; higher bond strengths favor $\mathcal{T} = 2$ at equilibrium, but may make the time it takes to reach equilibrium unacceptable at a particular temperature. The art and science of achieving $\mathcal{T} = 2$ conditions is well developed for periodic crystals [Lau70] but we must reproduce these results in physical systems in which we can implement our complicated self-assembly programs. Initial attempts at this are the subject of Chapters 3 and 4.

$\mathcal{T} = 1$ conditions, on the other hand, seem easier to implement — concentrations, for example can be as high as allowed by solubility. Further, we are free to pick temperatures and concentrations that optimize the speed of assembly. Given the difficulties associated with $\mathcal{T} = 2$, why not use $\mathcal{T} = 1$ self-assembly? Theoretically, what can $\mathcal{T} = 2$ assembly do that $\mathcal{T} = 1$ self-assembly cannot? Experimentally, are there difficulties with $\mathcal{T} = 1$?

The answers to these questions are not yet complete. Certainly, $\mathcal{T} = 1$ assembly cannot take advantage of cooperative binding — that trick for implementing functions and

constraining the order of growth is lost. One might thus suspect that universal computation at $\mathcal{T} = 1$ is impossible. In fact Len Adleman has demonstrated a construction for Turing machines (personal communication) at $\mathcal{T} = 1$. Given a Turing machine $\mathcal{M}$, Adleman's construction generates a tile system that produces a maximum terminal assembly if $\mathcal{M}$ halts; otherwise infinite assemblies are produced. Further the assembly has, written in tiles, the output of the Turing machine. Rather than depend on cooperative binding to implement the state-transition function of the Turing machine, Adleman's construction depends only on *geometry*; decisions are made based on whether self-assembly can progress beyond geometric obstacles in its path. Adleman guarantees that the obstacles are present in the assembly when needed by constructing his whole Turing machine simulation out of a single path of tiles that can grow only at the tip (in contrast to the "parallel" growth used in $\mathcal{T} = 2$ constructions like the counter). Thus $\mathcal{T} = 1$ assembly is capable of universal computation.

An interesting difference between Adleman's construction and the Turing machine construction in Figure 2.18 is that, at every step of the simulation, Adleman's construction generates a number of terminal assemblies that represent partial computations. Thus, rather than considering $Term(\mathbf{T})$ to be the output of the self-assembly program, Adleman considers only the largest element of $Term(\mathbf{T})$ to be the output — the rest of terminal assemblies are considered to be garbage. Adleman's construction makes a lot of these garbage assemblies, exponentially many in the length of time the Turing machine runs — it is horribly "assembly-inefficient". But even this can be overcome. Matt Cook has recently observed that a simple extension of Adleman's construction to 3D[41] works without

[41] In fact, Cook's construction requires only two layers of tiles.

generating garbage assemblies — if Turing machine $\mathcal{M}$ halts a unique terminal structure is produced.

Thus, using geometry alone, $\mathcal{T} = 1$ self-assembly exhibits rich theoretical behavior. In fact, proving theorems about $\mathcal{T} = 1$ seems harder and is in some ways more interesting than proving theorems for $\mathcal{T} = 2$ — embedding computational problems in $\mathcal{T} = 2$ assembly is easy and various complexity results follow; embedding computational problems in $\mathcal{T} = 1$ assembly is often difficult — proving when computational problems cannot be embedded in $\mathcal{T} = 1$ is *very* difficult — the power of geometry constantly surprises us. For example, it seems a very difficult problem to prove that the "comb" construction (Figure 2.12b) is the minimal way to construct a non-full square at $\mathcal{T} = 1$.

Experimentally, $\mathcal{T} = 2$ self-assembly still seems more promising than $\mathcal{T} = 1$. As far as implementing universal computation is concerned, we do not have a molecular self-assembly system[42] with rigid enough units to implement geometric tricks required for Adleman's construction. Even with such an experimental system in hand we would not compute using Turing machines at $\mathcal{T} = 1$ (nor would we do so at $\mathcal{T} = 2$); Turing machines are too slow a model of computation[43]. At $\mathcal{T} = 2$ the direct simulation of cellular automata, a much faster model of computation, is possible [Win96]; we have no $\mathcal{T} = 1$ construction for cellular automata or other fast models. Independent of computation, $\mathcal{T} = 1$ assembly does not appear to allow as tile-efficient constructions for patterns or shapes of interest; the case of program-size complexity for squares serves as an example. Still, the behavior

[42]The capillary force system in Chapter 3 *might* be rigid enough to create small assemblies. Assembling molecular tiles on a rigid 2D surface might also work ....

[43]We have not talked about the time complexity of self-assembly. Without doing so one can still see that the speed of a model of computation is relevant by assuming that if a model of computation requires more steps to compute a function, it must make an assembly containing proportionally more tiles.

of $\mathcal{T} = 1$ behavior is sufficiently rich that we should keep looking — perhaps there are interesting and efficient (fast and small) $\mathcal{T} = 1$ constructions yet to be discovered.

2.6.6 Singly Seeded versus Unseeded Assembly, Physics versus the TAM.

In this chapter we discussed the question of unique production for general tile systems, including those with more than one type of seed tile. When giving constructions, however, we restricted our attention to tile systems that are singly seeded — a single tile in the tile set was declared to be the seed and the produced assemblies[44] of the tile system were defined as exactly those assemblies that could be derived by the self-assembly relation $\rightarrow^*_{\mathbf{T}}$ from the seed tile. The produced assemblies are intended to be a prediction: we hope that if we make molecules corresponding to the tiles, then there exist conditions under which the molecules will form physical assemblies corresponding to the produced assemblies defined by the Tile Assembly Model. An astute reader may then ask, "How is a physical system to "know" which molecule is the seed tile? Why shouldn't the physical system use *any* of the molecules representing tiles as a seed? And if it does so, what keeps other $\mathcal{T}$-stable assemblies consistent with the tile set from forming? Then won't some of the singly seeded tile systems given here *fail* to uniquely produce?"

The short answer is that the physical system *doesn't "know"* which tile[45] is the seed tile; the physical system may use *any* tile as a seed tile. Also, it is likely that $\mathcal{T}$-stable assemblies other than those defined as produced assemblies *will form*, and it is likely that

[44] Of course here we mean the translation-invariant produced assemblies, $\widetilde{Prod}(\mathbf{T})$.
[45] I return to writing tile rather than molecule even where I mean "a molecule corresponding to a tile".

several of the singly seeded tile systems we give *will fail* to uniquely produce. Thus it seems that the physical world may naturally implement *unseeded tile systems* rather than seeded ones.

For a singly seeded tile system, it appears that the only way to guarantee that the physical assemblies produced correspond exactly to the produced assemblies is to *purify* the physical assemblies to get rid of assemblies that don't contain the seed tile. In the universe of purification techniques, such a simple purification is not very difficult. Given that we implement tiles as DNA, there exist many small functional groups (let us call them *tags*) that: (i) can be coupled to the DNA seed tile (ii) can also be bound reversibly to a second molecule, which we call an *affinity probe*. Such tags can be used in a couple of ways to purify the produced assemblies. For example, after the self-assembly is over, the assemblies can be poured over a solid support that bears the affinity probes. Only those assemblies containing seed tiles will bind[46] — other assemblies can be washed away — and the produced assemblies can be unbound and recovered from the affinity probes in a second wash under different conditions.

But we are mainly interested in autonomous self-assembly, without creating undesired structures that must be purified away. Even if purification is easy, undesired structures may waste large numbers of tiles. Thus we ask, for our singly seeded constructions, several questions: What structures different from the produced assemblies are likely to form? In general, do such structures violate the spirit of our constructions — that is, are they truly

[46] Approximately, anyway. Some assemblies without seed tiles will bond non-specifically and some assemblies with seed tiles will wash through, uncaptured. For typical commercially available purification methods, these events might occur with rates of less than 1% and less than 20% respectively. By chaining such separations together appropriately, in a manner similar to distillation, one can easily make these rates much smaller [RW99].

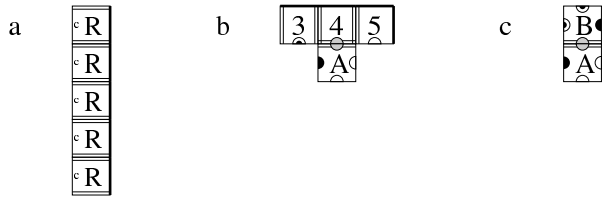


Figure 2.22: Physical systems are not guaranteed to produce exactly $\widetilde{Prod}(\mathbf{T})$ for singly seeded tile systems $\mathbf{T}$. By inspecting $\widetilde{Prod}(\mathbf{T_U})$ for the corresponding unseeded tile system $\mathbf{T_U}$ which we think is *more* physical, we find that additional assemblies may form. (a) The border tiles from the binary counter pattern may form arbitrarily long chains. (b) Tiles numbered other than '1' may initiate the assembly of the square construction from Figure 2.14b. (c) From the same construction tiles 'A' and 'B' can form terminal dimers.

undesired? If so, how can we minimize the formation of such structures so that *most* of the structures created are from the produced assemblies?

Let us refer to physically produced assemblies not in $\widetilde{Prod}(\mathbf{T})$ as *spurious assemblies*. Then, for the constructions we give, it seems that there are two types. First, there are spurious assemblies whose physical assembly seems logically inescapable, even in an ideal $\mathcal{T} = 2$ physical system — if our assumption is that bonds of strength-2 *can* form in a physical system, then these spurious assemblies *must* be able to form. Fortunately, for the constructions we give in this chapter, these spurious assemblies don't seem to violate the spirit of our constructions — they are subassemblies of the produced assemblies. Second, there are spurious assemblies whose assembly is more subtle — their assembly depends on the non-ideality of $\mathcal{T} = 2$ conditions in physical systems — for them it seems we may be able to find conditions that limit their occurrence. This is fortunate because these spurious assemblies are often unrelated to the produced assemblies; they are often truly *undesirable* spurious assemblies.

We consider first the "logically inescapable" spurious assemblies. For our singly seeded $\mathcal{T} = 2$ constructions $\mathbf{T}$, "logically inescapable" spurious assemblies are assemblies (larger than a single tile) that do not appear in $\widetilde{Prod}(\mathbf{T})$ but *do appear* in $\widetilde{Prod}(\mathbf{T_U})$ for the corresponding unseeded $\mathcal{T} = 2$ construction $\mathbf{T_U}$. Obviously, any such assemblies must contain at least one strength-2 bond. Recall the counter tiles from Figure 2.4, meant for use as a singly seeded tile system $\mathbf{T}$. Consider the unseeded $\mathcal{T} = 2$ tile system $\mathbf{T_U}$ that uses these tiles. Since, for such a tile system, the border tiles labeled 'L' and 'R' can serve as seed tiles *and* they can connect by strength-2 bonds, the produced assemblies $\widetilde{Prod}(\mathbf{T_U})$ will contain arbitrarily long chains of border tiles (Figure 2.22a) that do not contain the seed tile. Such chains of border tiles are, however, subassemblies of the infinite binary counter pattern — they can all self-assemble to the binary counter pattern.

Such assemblies are mildly "bad" in a couple senses: (i) they use up tiles without doing useful work (counting) and (ii) if such chains of border tiles are too numerous, it may be difficult to find assemblies bearing actual binary counters among them. In these respects the chains of border tiles are no different from many elements present in $\widetilde{Prod}(\mathbf{T})$ — arbitrarily large L-shaped assemblies containing only border tiles and the seed tile are included in $\widetilde{Prod}(\mathbf{T})$. In a physical experimental system, the solution to these problems is to tune the ratio of concentrations between the border tiles, seed tile, and rule tile so that, on average, the counter portion of the pattern grows about as fast as the borders [Win98a].

Our other finite constructions also have such "logically inescapable" spurious assemblies but they seem even less problematic. Consider the tiles that assemble the square in Figure 2.14b. The corresponding unseeded tile system might start assembling from the

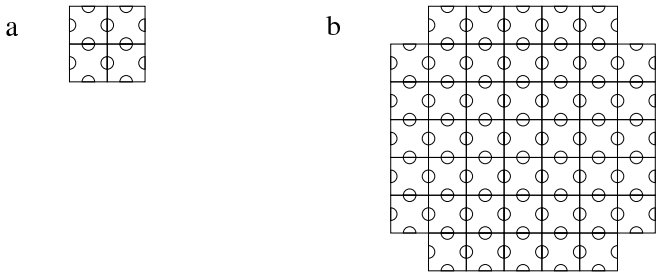


Figure 2.23: Physical systems are not guaranteed to produce exactly $\widetilde{Prod(\mathbf{T_U})}$ for unseeded tile systems $\mathbf{T_U}$. By considering that in physical systems $\mathcal{T} = 2$ is only an approximation, we find that additional assemblies may form. (a) Four copies of the blank tile from Figure 2.14b can form a 2-stable square, perhaps by the chance collision of all four tiles at once. This structure is still a subassembly of the desired assembly in Figure 2.14b and appears terminal. However, the assembly can enter new rows (or columns) via the transient binding of tiles by single strength-1 bonds; reinforced by the quick addition of a neighbor, the row may be quickly completed by the addition of tiles by pairs of strength-1 bonds. (b) By such mechanisms arbitrarily large assemblies of blank tiles (approximating squares given the right conditions) having nothing to do with the intended unique terminal assembly can be grown.

tile labeled '5' rather than the tile labeled '1' but it is easy to see that this causes no difficulties. Such structures (Figure 2.22b) are not terminal and can always be extended to form the full square. More interesting is the possible formation of *dimers* composed of the tile labeled 'A' and the tile labeled 'B'. Such dimers (Figure 2.22c) are terminal and cannot be extended by single tiles at $\mathcal{T} = 2$. However, such assemblies could be added, via pairs of strength-1 bonds, to structures that *can* be extended, for example Figure 2.22b. In a physical system we expect that such an addition of dimers is possible[47] even though such a mechanism is not part of the Tile Assembly Model. Thus even such small terminal dimers do not violate the spirit of our constructions — physically, a unique terminal structure may still be produced.

[47]Addition of dimers by a slightly different mechanism was observed to have high frequency in experiments described in Chapter 3.

More difficult to deal with than the "logically inescapable" spurious assemblies above, are $\mathcal{T}$-stable assemblies that are held together exclusively by strength-1 bonds. Consider the assembly of four "blank" tiles from Figure 2.14b depicted in Figure 2.23a. Such an assembly is, by our definitions, 2-stable; if it formed under $\mathcal{T} = 2$ conditions we believe that it would stick around. In physical systems, we believe that such structures do form: either they result from the chance collision of 4 single tiles at once[48] or from the transient binding of tiles by single strength-1 bonds. If we think we can create physical systems that have conditions approximating $\mathcal{T} = 2$ how can such single strength-1 bonds form? The answer lies in the way we believe physical systems can approximate the $\mathcal{T} = 2$ Tile Assembly Model.

In particular, rather forming bonds irreversibly as one might conclude from the Tile Assembly Model, physical systems form and *break* bonds. Further, bonds break and form at characteristic *rates*. If, as dictated by the rates, bond formation events exceed bond breaking events, assemblies grow, if vice versa, assemblies shrink. Given that there is no repulsion between mismatched bonds (see Section 2.6.4) we believe that single strength-1 bonds are formed (a $\mathcal{T} = 1$ event) at roughly the same rate[49] as strength-2 bonds or pairs of strength-1 bonds (both $\mathcal{T} = 2$ events). But we believe that isolated strength-1 bonds *break* at a higher rate than strength-2 bonds or pairs of strength-1 bonds. Thus, given appropriate rates for all of the events described, assemblies tend to grow by $\mathcal{T} = 2$ events since tiles added by $\mathcal{T} = 1$ events fall off quickly[50]. Occasionally, a tile bound by a single

[48] Such high order collision mechanisms become increasingly improbable very quickly as the number of tiles in a spurious structure increases; thus we don't worry about them.

[49] Given appropriate concentrations of tiles.

[50] This is, by the way, the mechanism by which cooperative binding arises; see Section 2.6.5 for the significance of cooperative binding.

strength-1 bond may be reinforced by the addition of an adjacent tile. In this way tiles added by $\mathcal{T} = 1$ events can be "frozen in" to an assembly.

This is the mechanism by which tiles like the blank tiles from Figure 2.14b can make arbitrarily large crystals (approximating squares in Figure 2.23b) in physical systems. It would seem that a square 2-stable assembly would be terminal. In physical systems, however, a new row of tiles is initiated not by a strength-2 bond as in our constructions, but by a transient strength-1 bond. A tile binding adjacent to the first tile reinforces it and the rest of the row can be filled in with $\mathcal{T} = 2$ events. At the same time tiles are falling off and occasionally an entire row disappears. This competition between shrinking and growing depends on the rates of additions, as dictated by tile concentrations. Above a certain threshold concentration, there is a particular size of square assembly, the *critical size*, above which assemblies are stable and grow, and below which they shrink. In a physical system it is only a matter of time before enough $\mathcal{T} = 1$ events conspire to create an assembly of the critical size — this is the *nucleation time*[51]. It turns out that the concentration above which we expect our $\mathcal{T} = 2$ constructions to grow is the *same threshold* above which arbitrarily large crystals of blank tiles are also stable and are expected to grow. Our constructions however, have short "nucleation times" since they depend on templates created using strength-2 bonds. Thus we are exploiting a race condition — we hope that our desired assemblies form and use up all the tiles before crystals like those based on the blank tiles (Figure 2.23b) have a chance to nucleate and grow. Arbitrarily close to the threshold, the nucleation time becomes arbitrarily long and, although the growth of our templated constructions becomes arbitrarily slow, it will have an advantage

[51] See [Lau70], Chapter 3 on the kinetics of crystal growth which includes a treatment of nucleation.

having started on our quick-nucleating templates. Thus in physical systems, we hope to create our desired structures, from the produced assemblies $\widetilde{Prod}(\mathbf{T_U})$ rather than spurious assemblies similar to the crystals of blank tiles.

The spurious assemblies we give as examples indicate that the Tile Assembly Model is by no means a complete model for physical self-assembly (not even for the limited domain of pseudo-crystalline self-assembly in which we are interested). Designing tile systems for uniquely produced assemblies is often made easier by considering only singly seeded tile systems. One can gain some insight into the physical behavior of such a tile system by taking the singly seeded tile system and examining its behavior as an unseeded tile system under the Tile Assembly Model. But, as in the case of the 'A-B' dimers, some spurious assemblies predicted for the unseeded system may not affect the tile system's ability to form a unique terminal structure. And some spurious assemblies that are undesirable, like those requiring occasional $\mathcal{T} = 1$ events are not detected even by considering the produced assemblies of an unseeded tile system. Thus, after using the Tile Assembly Model to guide the design of tile systems that are meant to be physically implemented, one must go outside of the Tile Assembly Model to more physical models of self-assembly to understand (i) whether structures not predicted by the Tile Assembly Model are likely to be produced and (ii) whether those structures do any real harm or actually just contribute alternative assembly pathways to the assemblies desired.

2.6.7 Unique versus Exact Production, Determinism versus Non.

It is clear that the algorithm (Theorem 7, p. 83) for deciding the unique production of a *ti*-assembly is just a variant of more general algorithms to decide exact production[52]. To make an algorithm that decides the exact production of a set of assemblies $\mathcal{A}$ we just throw out the preprocessing step that generates $\mathcal{A}$ from $\widetilde{A}$. To extend the algorithm to decide the exact production of a set of *ti*-assemblies $\widetilde{\mathcal{A}}$, we just run the preprocessing step on all *ti*-assemblies $\widetilde{A}$ in $\widetilde{\mathcal{A}}$, and take the union of the sets of assemblies that result as $\mathcal{A}$.

If it is so easy to decide exact production, why then don't we consider self-assembly programs that exactly produce rather than uniquely produce? In doing so, could we get better bounds for program-size complexity? At their core these questions point to a single question: Of what use is non-determinism in self-assembly? In this chapter we avoid largely avoid non-determinism. Only a single one of our constructions, the "comb" from Figure 2.12b with tile '5' as a seed tile, makes use of non-determinism but it uniquely produces a single *ti*-assembly. This stimulates the question: Is there ever a reason to exactly produce a set of inequivalent *ti*-assemblies?

The most obvious use for inequivalent *ti*-assemblies is non-deterministic computation: we may wish to simulate non-deterministic Turing machines or cellular automata by growing a *ti*-assembly corresponding to each computational trajectory *in parallel*. This paradigm for computation by self-assembly is already being explored[53] — theoretically for the purpose of solving 3-SAT [LL00] and experimentally for the XOR of short bit

[52] A fact pointed out to me by Matt Cook.

[53] Actually, Adleman's original construction for the Directed Hamiltonian Path Problem, [Adl94], was the first exploration of this paradigm for *linear* self-assembly. The first step of his molecular algorithm uses linear self-assembly to non-deterministically sample paths from a graph.

strings [LWR00, MLRS00]. Such constructions are very similar to our deterministic-RC constructions; they are essentially "non-deterministic-RC" and produce families of related RC assemblies. Not surprisingly, proving the correctness of such constructions[54] is, as it is for deterministic-RC assemblies, very easy. Thus, in a sense, such constructions use non-determinism in a simple way. More complicated non-deterministic self-assembly programs might use non-determinism in ways that require the general algorithm for deciding exact production to verify their correctness. Whether such self-assembly programs would be useful or interesting remains an open question.

2.7 Coda

There are many other opportunities for studying self-assembly (and chemistry in general) from the point of view of computational complexity. In addition to program-size, standard complexity measures in computer science include time, space, and decidability. For self-assembly too, many natural complexity measures can be defined. These include the aforementioned program-size complexity (in chemical terms, the number of types of reactant molecules), as well as time complexity (in chemical terms, the kinetics of self-assembly), decidability of the output set (in chemical terms, the complexity of the product molecules), and growth rule complexity (in chemical terms, the mechanism of self-assembly). While these complexity measures have been studied by chemists for years, they have been little studied with the machinery of computer science — we are aware of only a handful of papers that do so. For time complexity, Adleman has proposed a model that emphasizes counting discrete time-steps. Application of his model to the self-assembly

[54] That is, that they exactly produce the set of all valid computations.

of N-long linear polymers [Adl00] yields a time complexity of $O(N \log N)$, a result that appears related to the problem of sorting. Others, with Adleman [ACGH01], have found the time complexity of self-assembling $N \times N$ squares — they give constructions that take advantage of the parallel nature of self-assembly and yield time $O(N)$ where $O(N^2)$ might have been expected. The decidability of set product molecules (as a function of the complexity of reactant molecules) has been studied for the self-assembly of linear and branched DNA molecules by Erik Winfree [Win98a] — interestingly, it recapitulates the Chomsky hierarchy. The complexity of growth rules has been examined, albeit implicitly, in the study of Penrose tilings and quasicrystals (see Chapter 3, section 3.5 for a brief discussion of this) — it raises the issue of whether there might exist certain arrangements of molecules that cannot, in principle, be self-assembled. These examples hint at the promise of studying computational complexities for self-assembly; much work remains to be done.

Chapter 3

Using lateral capillary forces to compute by self-assembly

Surface tension binds all things ...

—*Master Yoda from* The Detergent Strikes Back[1]

3.1 Abstract

Investigations of DNA computing have highlighted a fundamental connection between self-assembly and computation: in principle, any computation can be performed by a suitable self-assembling system. In practice, exploration of this connection is limited by our ability to control the geometry and specificity of binding interactions. Recently, a system has been developed that uses capillary forces[2] to assemble plastic tiles according to shape complementarity and likeness of wetting [BTCW97]. Here[3] the capacity of this system to compute by self-assembly is explored. Tiles were prepared to test the system's ability to generate three structures of increasing complexity: a periodic "checkerboard"

[1]This was fabricated by me.

[2]Here capillary force is a synonym for interfacial tension which, when air is involved as one of the phases, goes under a more familiar moniker, *surface tension*. If these terms are opaque, read Appendix B.

[3]Most of the material in this chapter draws from [Rot00]. See Chapter 1 Section 1.6 for details.

tiling, an aperiodic Penrose tiling, and a "computational"[4] tiling that simulates a one-dimensional cellular automaton. Matching rules for these tilings were enforced by coating tiles with patterns of hydrophobic and hydrophilic patches or "wetting code". Energetic, kinetic and mechanistic details of self-assembly explain differences between experimental structures and mathematically ideal ones. In particular, the growth mechanism observed appears incompatible with computations that make use of a chosen input.

3.2 Introduction

Self-assembly and computation are linked by the study of mathematical tiling [GS87]. A tiling is an arrangement of tiles (shapes) that covers the plane. Tiles fit together according to matching rules: their edges must have complementary shapes and must agree on additional markings such as colors. Given any computing device (*e.g.* a Turing machine [Wan63], or cellular automaton) tiles and matching rules can be designed so that the tilings formed correspond to a simulation of that device (typically, successive rows in the tiling represent the memory of the device at successive timesteps). Computation by tiling is thus universal.

Tiles, matching rules, and the tilings they define are abstract mathematical objects but their geometrical natures suggest physical analogues. Real objects (*e.g.* atoms) may be thought of as tiles, binding interactions (*e.g.* chemical bonds) as matching rules, and

[4]The structure created by any set of tiles can be interpreted as a computation; so too can the operation of any machine. Still, one neither refers to all tile sets as computational nor calls all machines computers. Here the adjective computational, like the term computer itself, connotes intent to perform a computation of interest, or of particular power.

self-assembled structures (*e.g.* molecules) as partial tilings. Such an analogy has been used to describe the structure of quasicrystals in terms of Penrose tilings [LS84].

Together, the connection of computation to mathematical tiling and analogies between tiling and self-assembly imply that, in principle, any computation can be realized as a self-assembled structure. Conversely, the Church-Turing thesis implies that all self-assembled structures can be viewed as computations. Why study such connections? Originally, in the context of DNA computation, the purpose was to perform computation and compete with electronic computers. DNA self-assembly has been used in the solution of an NP-complete problem [Adl94], and to create 2-D DNA lattices [WLWS98a] as a step toward simulating cellular automata [WYS98]. Competing with electronic computers is a difficult goal, one that seems unlikely to be met. Rather, the link between computation and self-assembly may find most use in defining the structures accessible by self-assembly, both theoretically and practically. Theoretically because the mathematics of computation may be used to classify self-assembled structures (*e.g.* to map self-assembled structures onto languages in the Chomsky hierarchy, [WYS98]) or to analyze the resources, in terms of time or tile complexity, needed to create a particular structure (*e.g.* to find the smallest number of tile species that assemble uniquely into an $n \times n$ square [RW00], see Chapter 2). Practically because tilings that encode computations provide new synthetic targets — structures more complex, in general, than any considered by chemists so far. Often they are nonperiodic and require many distinct binding interactions. Thus assembly of "computational" tile sets may test the limits of synthesis by self-assembly. Here such practical issues are explored. The smallest number of binding interactions that allows universal computation is not known, but straightforward encodings of many computations require a dozen or more

(*e.g.* enumerating primes, [GS87]) suggesting DNA as the best medium for making such structures. Recently, capillary force based systems [HSM96, BTCW97, BCGW99] have become alternatives. Like DNA, these systems allow many binding interactions; unlike DNA, they allow easy control of tile geometry and visual inspection of reaction mechanism. Capillary force self-assembly operates on particles as small as 55 nm [DDYN94] and proteins a few nm in size appear to assemble by capillary forces in lipid bilayers [KPDN95]. Thus capillary force self-assembly may allow microfabrication of computational tilings. Here I show that such a system is rich enough to enforce four matching rules — sufficiently many to make tiles encoding a simple binary function, exclusive-or (XOR), and simulate a 1-D cellular automaton to create triangular patterns.

Unfortunately, simply creating binding interactions to enforce matching rules does not ensure that tiles assemble as desired. Antithetical to most chemists' idea of good rational design, distinct tiles may bear the same matching rule and compete for a single lattice site. I term a tile set "competitive" if it can form a lattice site such that one tile is "correct" (matching the site on all sides), and another tile competes and is "incorrect" (matching and mismatching the site on at least one side). Natural encodings of computations as tiles often yield competitive tile sets [GS87] and an important open question is: Are competitive tile sets are required for universal computation? For such tiles, error-free assembly requires that binding be cooperative. I define binding to be cooperative if the equilibrium constant for association of a tile at a site increases strictly with the number of bonds the tile can make with that site. For experiments reported here cooperative binding appears responsible for an error rate ≈ 6 fold lower than noncooperative binding would predict. Finally, for many computational tile sets, specifying a particular computation

requires that tiles bind into corner sites on a special "input tile". Here tiles are observed to bind by a mechanism that precludes computation using such a chosen input.

3.3 Materials and Methods

3.3.1 Ideal structures and simulations.

Wulff's rule ([Mar95], pp. 13–20, [LNP92], pp. 269–273) gives square ideal structures for checkerboard or XOR tiles and a decagonal ideal structure [GL87] for Penrose tiles [Gar77, Pen78]. The Penrose tile ideal structure given is approximate. Simulations were initialized by assigning a random position to each tile (168 checkerboard tiles, 420 Penrose tiles, 256 XOR tiles) in a square space (discrete for checkerboard and XOR tiles, continuous for Penrose tiles) of a size (25, 31, or 27 tile widths wide, respectively.) chosen to match the dimensions of the self-assembly experiments described below. Each step, one structure (collection of one or more tiles connected by bonds) was chosen at random and moved one tile width in a random direction (up, down, right, or left) if no overlap occurred with other tiles. (Penrose tiles were moved with a displacement chosen randomly from [0,2) and a 66% chance of rotation $\pm 36°$.) Any collision of the moved structure with stationary ones for which the edges involved satisfied a matching rule caused the creation of a bond. Each simulation continued until all tiles joined a single structure or 2000 moves passed without formation of a bond. On average ($n = 40$), simulations terminated in $\approx$ 1400, 6000, or 900 steps leaving 1.1, 5.2, or 1.1 structures for checkerboard, Penrose, or XOR tiles, respectively.

3.3.2 Preparation of tiles.

All tiles were laser cut (ULS 25, Universal Laser Systems) from 3 mm fluorescent yellow cast acrylic (Solter Plastics, here it appears light gray in color). Before cutting, the bottom of the acrylic was made hydrophilic by wetting it with cyanoacrylate ester (Zap-O, Pacer Technology) and dusting with silica gel (type G, Sigma) three times. The top of the acrylic was sprayed fluorescent blue (which appears here as dark gray). Patterns indicating matching rules were created by burning away paint at a low power setting. Notches (1 mm deep, .2 mm wide) were cut between regions to be made hydrophobic or hydrophilic. Tiles were cut from the acrylic sheet at a high power setting, stacked, and covered in tape. Using the notches as guides, tape was cut away from regions to be made hydrophilic. The resulting stacks of tiles were wet with cyanoacrylate and dusted with silica three times yielding a layer of cyanoacrylate/silica $\approx$ .1 mm thick. Hydrophobic patches were deprotected and painted with fluid (naptha, butadiene rubber, butyl Cellosolve) from an hydrophobic slide marker (PAP Pen, Research Products International) diluted $7:1$ in hexane.

3.3.3 Visualization of the meniscus.

Hydrophobic sand (CENCO, Central Scientific Co.) was sprayed with red fluorescent paint (here it appears light gray). Visualization of hydrophilic patches was performed by adding 30 mg sand/cm^2 interface between mineral oil (Aldrich, 0.83 g/ml) and aqueous sodium metatungstate (Geoliquids, 1.65 g/ml).

3.3.4 Self-assembly.

All experiments used an n-hexadecane (Sigma, 0.77 g/ml) superphase and an aqueous sodium metatungstate subphase. Tiles were shaken with an AROS 160 shaker (Barnstead-Thermolyne, 3.8 cm orbit). Before use, tiles were soaked 30 min in water. The tiles in Figure 3.2 were placed in a 10 cm square box with 100 ml subphase and 50 ml of superphase, disaggregated with forceps while shaken at 78 rpm so that $\approx 50\%$ of bonds were broken, and equilibrated at 75 rpm. The ratio of hydrophilic/hydrophobic bonds was stable after 30 s but data were taken after 5 min. Checkerboard tiles were shaken in an 18 cm square glass dish with 120 ml superphase and 120 ml subphase (1.65 g/ml) for: 5 min at 44.5 rpm, 15 min at 40.9 rpm, 45 min at 37.6 rpm, 135 min at 34.0 rpm, 405 min at 30.6 rpm, and 1215 min at 27.0 rpm. Penrose tiles were shaken in a 38 cm square glass box with 450 ml superphase and 1100 ml subphase (1.65 g/ml) for: 3 h at 22.6 rpm, 6 h at 21.8 rpm, 12 h at 21.1 rpm, 24 h at 20.5 rpm, 48 h at 19.8 rpm, 48 h at 19.0 rpm, 24 h at 18.3 rpm, 24 h at 17.6 rpm, and 24 h at 16.9 rpm. XOR tiles were shaken in a 32 cm square glass box with 220 ml superphase and 860 ml subphase (1.65 g/ml) for: 12 h at 23.7 rpm, 24 h at 21.8 rpm, and 24 h at 18.3 rpm.

3.4 Results and Discussion

3.4.1 Ideal structures, simulations, and prerequisites for computation.

Modeling suggests (i) under a given set of growth rules for self-assembly, certain tile sets are particularly prone to form defects (ii) for a particular tile set different growth rules yield structures of very different quality. This raises two questions: (i) Will tile sets

that encode computations be, in general, prone to defect formation? (ii) If so, what growth rules must self-assembly follow so that such tile sets self-assemble successfully? To explore these questions I compare the hypothetical and experimental self-assembly of a competitive set of computational tiles (XOR tiles, Figure 3.1I) with two simpler sets of tiles: "checkerboard" tiles (Figure 3.1A), and Penrose tiles, (Figure 3.1E). XOR tiles were studied since competitive tile sets with the same form (two inputs and a one duplicated output) exist that are universal (from the universality of one-way CA, [Mor92]). Thus if conditions for correct XOR tile assembly can be found, any computation can be performed.

For each set of tiles an "ideal structure" is given (Figure 3.1B, F, and J), intuitively an example of the best one could expect self-assembly to produce. An ideal structure is, for a fixed number of tiles, one of many structures that minimize unmatched edges and violate no matching rules. For comparison, structures (Figure 3.1C, G, and K) were created by simulation using a model of self-assembly [Mea83, KBJ83] where binding is irreversible[5]. These structures are far from ideal: dendritic rather than compact, having only $\approx 60\%$ of the bonds (matched edge pairs) present in the ideal structures, and bearing vacancy defects and "errors" (sites where abutting edges mismatch). Strikingly, error rates range over almost two orders of magnitude: 0.2%, 2.2%, and 14.7% for checkerboard, Penrose, and XOR tiles resp. (%error = mismatches/[mismatches + bonds]).

Diagrams of bond formation (Figure 3.1D, H, L) contrast error-formation mechanisms for the three tile sets. Only mechanisms involving monomers are shown, but analogous mechanisms involving multi-tile structures also occur. Checkerboard tiles made errors only

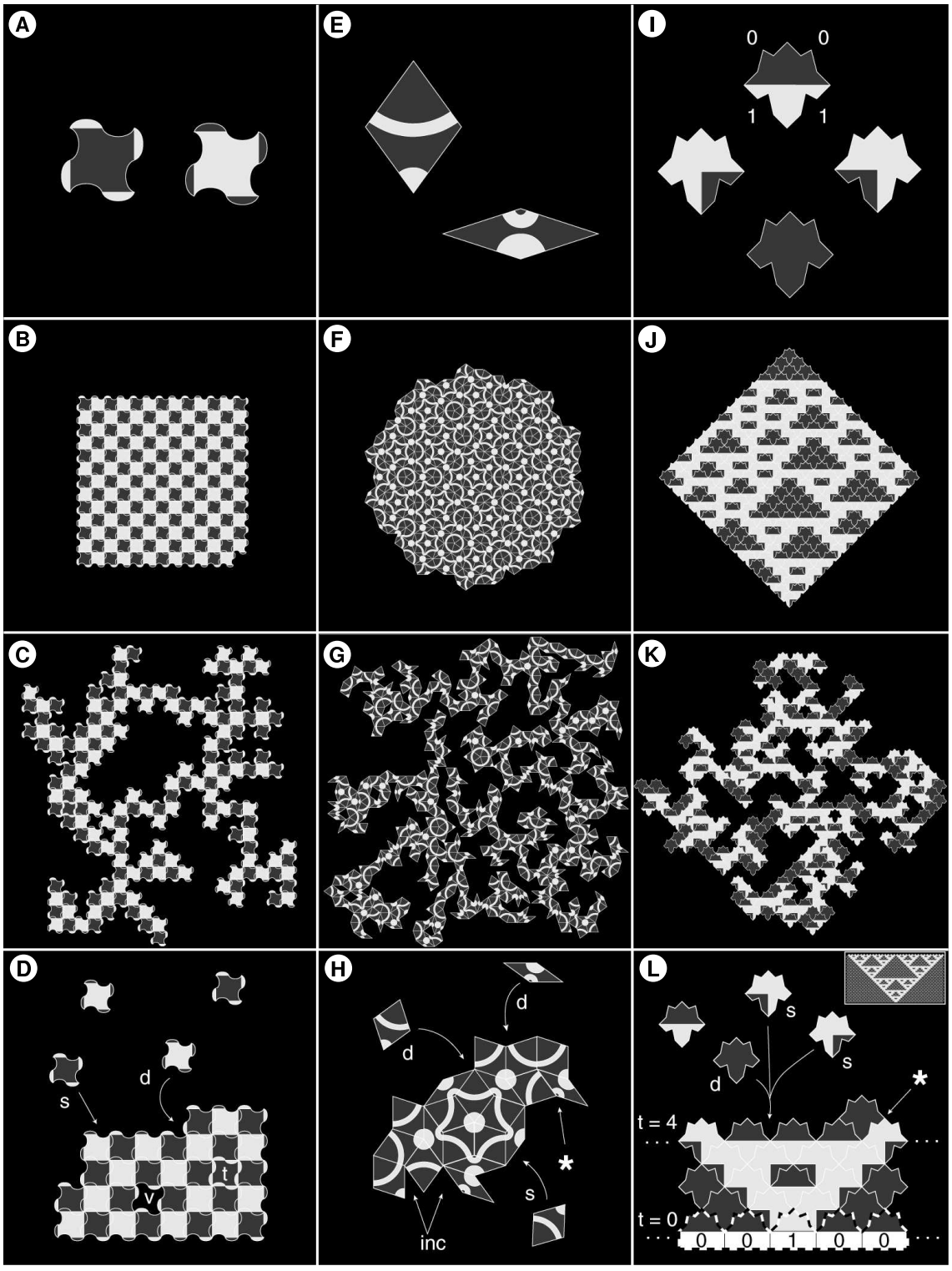
[5]This model of self-assembly is closely related to the $\mathcal{T} = 1$ model of self-assembly presented in Chapter 2.

when a structure was mechanically trapped in a larger structure (Figure 3.1D, tile t); no bonds held such a tile in the lattice. No Penrose tiles and few XOR tiles were trapped; for them errors (100%, $>$ 99%) resulted from association of a tile (or larger structure) matching the lattice on one edge but mismatching on another (Figure 3.1H and L, $\star$). Penrose tiles made such errors when "incompletable" sites (for which no tile with correct matching rules for both sides of the site exists, Figure 3.1H, inc) were filled. XOR tiles did form and fill incompletable sites but many errors formed since XOR tiles are competitive: at a single site a correct tile (Figure 3.1L, d) that binds by two bonds competes with two incorrect tiles (Figure 3.1L, s) that bind by single bonds.

A tendency for Penrose tiles [Els85] and tiles similar to XOR tiles [Rei99] to form defects has been noted before and raises questions of whether (i) self-assembly of Penrose tiles models the formation of well-ordered quasicrystals and (ii) self-assembly of XOR tiles can perform error-free computation. In fact, the self-assembly of Penrose and XOR tiles is highly growth rule dependent. Conditions for nearly defect free assembly of Penrose tiles are complex and are treated elsewhere [OSDS88, DiV90, RBJ90]; we explain briefly how Penrose tiles are special in section 3.5.

To understand conditions sufficient for XOR tiles to assemble without defects and compute as desired, consider first how they compute (see legend for Figure 3.1I and L). If tiles are labeled 0 or 1 based on the identity of their output bits, a row of tiles can be read as a string of binary digits that represent the memory cells of a 1-D blocked cellular automaton (BCA) at a single timestep in its simulation. The next row (immediately above)

Figure 3.1: Tiles, ideal structures, and structures formed by simulation. (A) Checkerboard tiles. (B) Periodic ideal structure with a minimal number of unmatched edges. (C) Simulation results. Few errors (0.2% of abutting edges are mismatched, $n = 40$) but many unbound edges are present. (D) Schematic shows tiles binding by a single (s) bond along a face or by a double (d) bond into a corner. Errors form only by mechanical trapping (t). Vacancies (v) also form. (E) Penrose tiles. (F) Aperiodic ideal structure (approximate). (G) Simulation results. A moderate number of errors (2.2%) and three separate structures are present. (H) Schematic shows tiles binding by single (s) or double (d) bonds. Matching rules allow incompletable sites (inc) into which no tile can fit without error ($\star$) which occurs when such a site is filled. (I) XOR tiles encode the binary function XOR ($X \oplus Y = Z$). Concave lower edges denote a pair of input bits X and Y (light gray = 1, dark gray = 0). Convex upper edges denote two copies of an output bit Z. The top tile encodes $1 \oplus 1 = 0$, the leftmost tile $1 \oplus 0 = 1$, the rightmost tile $0 \oplus 1 = 1$ and the bottom tile $0 \oplus 0 = 0$. (J) Ideal structure. (K) Simulation results (rotated 45°for viewing convenience). Many errors (14.7%) are present. (L) Schematic shows XOR tiles assembling on an input tile, outlined in by a dotted line, denoting the string "00100" continued on both sides with zeros. Successive rows correspond to successive timesteps ($t = 0, \ldots, t = 4$) in a simulation of the corresponding 1-D BCA ([Wol94], rule 60). Arrow indicates three tiles competing for a corner site, two of which (s) would form an error ($\star$) if incorporated. Inset: a larger, perfect section of the intended structure, Pascal's triangle mod 2.


A
E
I
0
0
1
1
B
F
J
C
G
K
D
s
d
V
t
H
d
d
*
s
inc
L
s
s
d
*
t = 4
t = 0
0 0 1 0 0

represents the memory cells in the next timestep where each memory cell in the next row is the XOR of two neighboring cells. The ideal structure in Figure 3.1J is then a fragment of a "random" computation, the result of simulating the BCA on a random binary string. Figure 3.1L shows the beginning of a specific computation where the assembly of XOR tiles has been nucleated on an "input tile", a long special tile that provides a chosen input to the computation. In this case XOR tiles would form Pascal's triangle mod 2 (inset Figure 3.1L), an example where computational tiles provide a striking synthetic target — a discrete approximation of the Sierpinski triangle fractal. Without such nucleation, all possible computations are created and a particular computation of width n would be just one of 2^n competing structures similar to Figure 3.1J. Trivially, to achieve such nucleation, tiles must bind corners (sites comprising a pair of perpendicular edges). Additionally, assuming that tiles can bind to a lattice via a single bond: (i) Tiles must bind reversibly so that vacancies, incompletable sites, and errors can be resolved. (ii) Associations must be cooperative (a tile that matches a corner on both edges must be preferred over a tile that matches on one edge) so that errors are infrequent. Two types of incorrect tile compete with one type of correct tile for a corner site; without cooperative binding each would bind (and dissociate) equally often yielding an error rate of 1/3 (33%). With cooperative binding correct tiles dissociate at a lower rate than incorrect tiles; if the strength of a single bond is $\Delta G < 0$, the preference of a correct tile over an incorrect one at equilibrium is $e^{-\Delta G/RT}$ [Win98a] so, in principle, very low error rates can be achieved.

3.4.2 Physical implementation.

To create real tiles analogous to those in Figure 3.1, the system described in [BTCW97] was used. Millimeter-scale tiles floating between a hydrophobic and a hydrophilic phase deform the interface by trapping it above or below its equilibrium height. This happens wherever the side of a tile with hydrophobic character protrudes into the hydrophilic phase and vice versa (Figure 3.2A–C). Capillary forces between tiles are attractive or repulsive, respectively, if movement of the tiles decreases or increases the area, and hence the energy, of the interface [KN94]. When a vessel containing such a system is shaken on a rotary shaker, tiles with strong interactions form and maintain stable associations termed "bonds". Binding is reversible; shaking at higher frequencies causes bond equilibria to shift toward dissociation. Thus shaking is an analogue for temperature. Tiles were shaken with a decreasing frequency profile to anneal structures toward the ideal structures given in Figure 3.1.

Enforcing the matching rules given in Figure 3.1 requires the use of up to four (for XOR tiles) different binding interactions that must be specific and isoenergetic. Specificity may be conferred by giving tiles complementary shapes or patterns of wetting [BTCW97]. Attempts to make binding interactions based entirely on shape failed (not shown) since noncomplementary shapes had binding energies too similar to complementary ones. To overcome this I introduced patterns of alternating hydrophilic and hydrophobic patches or "wetting code" to enforce matching rules and made limited use of shape.

The design of good wetting codes is subject to constraints that derive from both the logic of the matching rules and the physics of the system. Conceptually, it was useful to view wetting codes as binary strings. The Hamming distance (number of mismatches)

between pairs of codes could then be optimized. Wherever possible, Hamming distances for mismatches were made greater than half the length of the code. This insured that mismatches were actually repulsive. Self-interactions were avoided by making wetting codes nonpalindromic. Finally, for wetting codes applied to straight edges, care was taken to minimize shifted alignments of the edges with many unintended matches in wetting character.

Yet physics dictates that one cannot treat hydrophilic and hydrophobic patches merely as zeros and ones. An uneven distribution of hydrophilic and hydrophobic patches around the perimeter of a tile may cause it to tilt [BCGW99] and inhibit aggregation. Further, the degree of deformation of the interface along an edge (and hence bond strength) is not simply a function of the number of hydrophobic and hydrophilic patches. A sequence of patches induces a wave-like meniscus that changes curvature at each change in wetting character. The amplitude of this wave at a given patch is roughly the width of a patch [GN96] and is limited by the thickness of the tile. Consecutive patches of the same type are, effectively, part of one large patch. Thus a sequence hydrophobic, hydrophobic, hydrophilic, hydrophilic does not change the interfacial area in the same way as an hydrophobic, hydrophilic, hydrophobic, hydrophilic sequence. The use of sequences with a similar number and distribution of changes in wetting character should help minimize differences in meniscus deformation and create isoenergetic bonds. Penrose tiles were not designed weighing this criterion heavily and the consequences are described later.

Another way to regularize the contribution of a patch to the interfacial tension is to equalize the strength of hydrophilic and hydrophobic interactions. Figure 3.2A–C diagram, in cross section, hydrophobic and hydrophilic interactions between tiles for three

different equilibrium positions at the interface between a hydrophobic superphase and a hydrophilic subphase.[6] The strength of a bond between two tiles is proportional to the change in area of the meniscus if the two tiles are brought from an infinite separation into contact; the area of the meniscus is proportional to the length of the meniscus, as drawn in cross section. In the previously described water/perfluorodecalin system [BTCW97] polydimethylsiloxane tiles float so that most of their thickness lies in the water superphase (upside down, but analogous to Figure 3.2A). Thus in this system, hydrophobic bonds are strong and hydrophilic bonds are weak. In the system described here, acrylic tiles float between hexadecane and a sodium metatungstate solution. By changing the concentration of salt in the subphase, the equilibrium position of the tiles at the interface was adjusted. This allowed the strengths of hydrophilic and hydrophobic bonds to be equalized and a repulsive force between hydrophobic and hydrophilic patches to be maintained (Figure 3.2B).

Hydrophilic and hydrophobic bonds are equally strong when the tiles sit midway between the two phases. At this midway point, assuming that a tile has an equal number of hydrophilic and hydrophobic patches of similar geometry, the capillary forces' contribution to its position cancels. Then the equilibrium position of the tile depends only on gravity and buoyancy and we consider the position of the tile as if there were no interfacial tension and the meniscus were flat (Figure 3.2D).[7] The force due to gravity is just

[6]Tiles of the same density but lying either wholly in a hydrophobic subphase, or wholly in a hydrophilic subphase do not, in general, have the same equilibrium position as is shown in Figure 3.2A–C; the capillary forces they feel push them in opposite directions and the buoyant forces on them are different. Assume instead that the tiles drawn have differing densities that place them at the same equilibrium position or, better still, that the tiles are like those of Figure 3.2E, and have a symmetry in the undepicted dimension that guarantees they have the same equilibrium position.

[7]It is clear that away from the midway point, the capillary force cannot be neglected. Tiles with a density of 1.21 g/ml can be made to float metastably on water subphase of density 1.0 g/ml (as can a

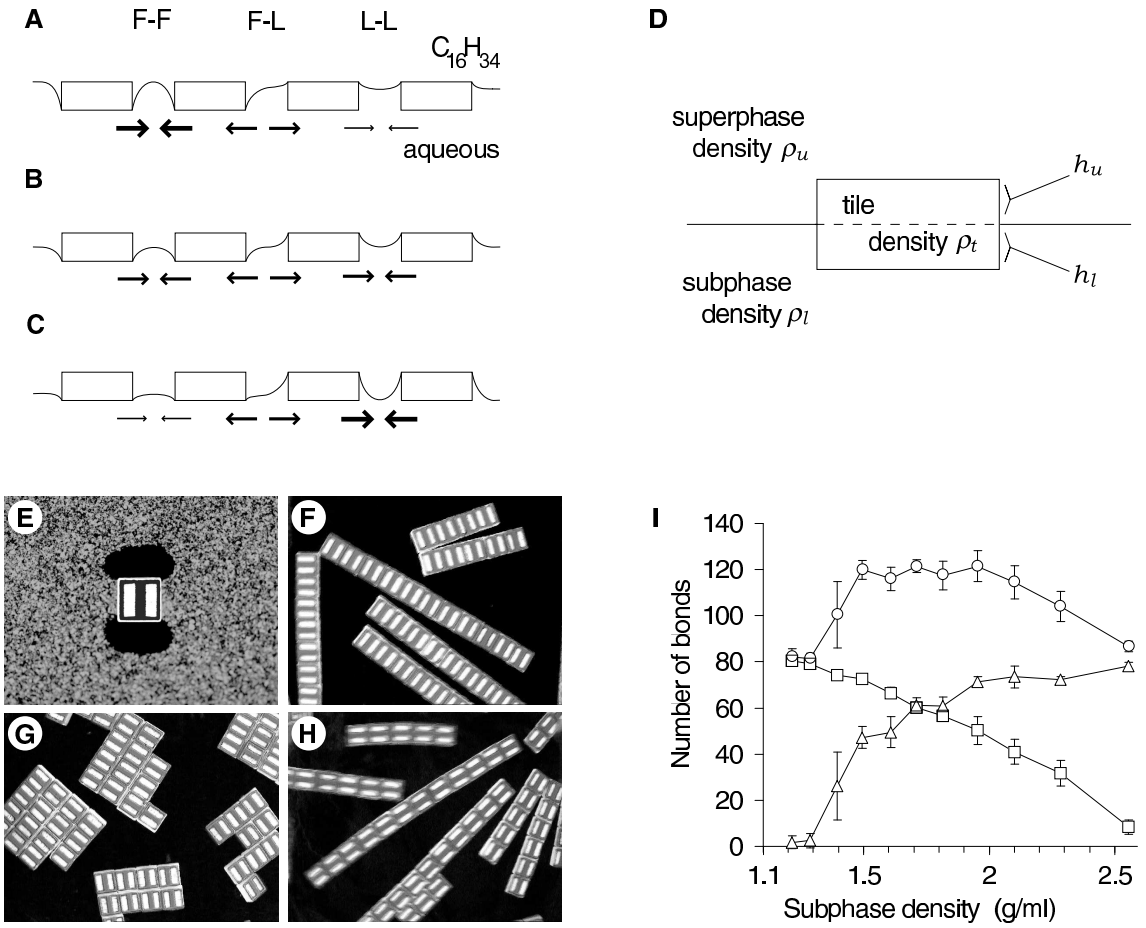


Figure 3.2: Changes in self-assembly as a function of subphase density. All subphase densities ρ, given in g/ml. (A–C) Side views diagram menisci between tiles at the interface of a hexadecane superphase ($C_{16}H_{34}$) and aqueous sodium metatungstate subphases (aqueous) of increasing density. Tiles: (A) sink deeply into a dilute subphase, (B) rest halfway between phases on a more concentrated subphase, (C) ride high on a saturated subphase. F-F, F-L, and L-L indicate the meniscus between hydrophobic (F) and/or hydrophilic (L) tiles. Tiles move to minimize the area, and hence energy, of the interface. Arrows give the direction and, for attraction, relative strength of the force between tiles. (D) Diagram used to derive the subphase density for which hydrophilic and hydrophobic interactions have equal strengths. (E) Top view of a 5mm square tile with one pair of opposing faces made hydrophilic and the other made hydrophobic. The interface has been doped with hydrophobic sand that moves away from hydrophilic sides and towards hydrophobic sides to show lines of constant height in the meniscus. (F–H) Assembly under conditions analogous to (A–C) with $\rho = 1.3$ (F), 1.71 (G), or 2.4 (H). Note different tile orientations in (F) and (H). (I) Number of hydrophilic and hydrophobic bonds plotted as a function of ρ. Error bars are $\pm$ 1 s.d. hydrophilic bonds (Δ) dominate for $\rho > 1.71$; hydrophobic bonds ($\square$) dominate for $\rho < 1.71$. The total number of bonds ($\circ$) is roughly constant from $\rho = 1.49 - 1.95$. Bond strengths are equal for $\rho = 1.71$, close to the density ($\rho = 1.65$) predicted.

$F = Mg = \rho_t A(h_u + h_l)g$ where M is the mass of the tile, g is the gravitational constant, A is the area of the tile, ρ_t is the density of the tile, and h_u and h_l are the thickness of the tile above and below the interface, respectively. Archimedes' law states that the buoyant force on an object displacing a volume V of fluid with density ρ is just $F = \rho V g$. Given a density ρ_u for the superphase and ρ_l for the subphase, the upper part of the tile generates a buoyant force $\rho_u A h_u g$ and the lower part of the tile generates a buoyant force $\rho_l A h_l g$. Equating the downward force due to gravity with the upward buoyant forces we get:

$$\rho_t A(h_u + h_l)g = \rho_u A h_u g + \rho_l A h_l g$$

Cancelling As and gs and rearranging yields:

$$\frac{h_u}{h_l} = \frac{\rho_l - \rho_t}{\rho_u - \rho_l} \tag{3.1}$$

When the tile sits midway between the phases the ratio h_u/h_l is equal to 1. Thus, setting the right hand side of Equation 3.1 equal to 1, setting $p_u = .77$ g/ml for the hexadecane superphase, and setting $\rho_t = 1.21$ g/ml for an acrylic tile yields a density for the aqueous subphase of $\rho_l = 1.65$ g/ml — a density for which the strengths of hydrophilic and hydrophobic bonds should be equal.

To test this, 5-mm square tiles were made so that one pair of opposing sides was hydrophilic, and the other hydrophobic as visualized in Figure 3.2E. Tiles formed linear structures with hydrophobic bonds when shaken at low subphase densities (Figure 3.2F,

paperclip with a density around 8 g/ml). In this situation, where gravity overwhelms buoyancy (or in the opposite situation for objects less dense than the superphase), it is capillary forces that hold the tiles at the interface.

$\rho = 1.3$ g/ml), square structures with hydrophobic and hydrophilic bonds at intermediate densities (Figure 3.2G, $\rho = 1.71$ g/ml), or linear structures with hydrophilic bonds at high densities (Figure 3.2H, $\rho = 2.4$ g/ml). The relative strength of the hydrophobic and hydrophilic bonds were measured (Figure 3.2I) by shaking the tiles at a constant rate until the structures formed were in equilibrium and then counting hydrophobic and hydrophilic bonds. At the subphase density for which these numbers were equal — 1.71 g/ml — the bond strengths were equal, in good agreement with prediction.

3.4.3 Self-assembly.

To show the assembly of checkerboard tiles using a simple wetting code, two kinds of square tiles were prepared. The side of each tile bore a two-patch code, for which half of each side was made hydrophobic and the other hydrophilic (Figure 3.3A). Tiles were disaggregated (Figure 3.3B) and shaken for ≈ 30 h, after which structures (Figure 3.3C) with a few vacancies were observed. The structures had 288 ± 8 bonds ($n = 4$) — 92% of the 312 bonds in an approximately square ideal structure. Figure 3.3D shows "shifted" bonds — artifacts of the chosen wetting code common before annealing (Figure 3.3B), but rare once annealing began.

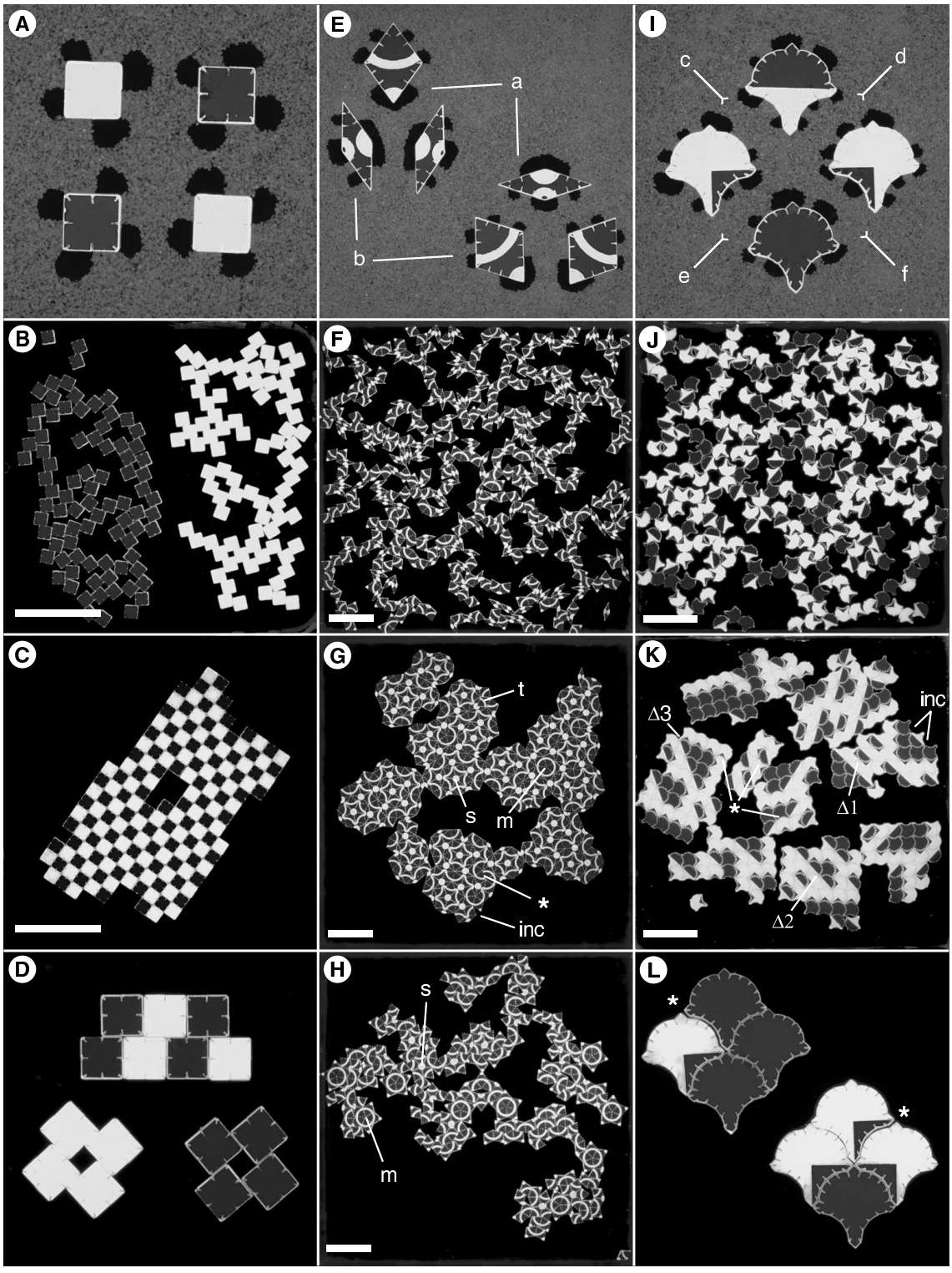
Penrose tile matching rules require two kinds of nonpalindromic bonds. To enforce them a four-patch wetting code (Figure 3.3E) was used. Each rhomb was coated on two sides with a "type-a" code and on two sides with a "type-b" code (Figure 3.3E, a and b). Rhombs were mixed in a ratio of $\tau : 1$ (fat : thin, $\tau = 1.618\ldots$, the golden mean) and disaggregated until no complete vertex star (arrangement of tiles around a vertex) occurred (Figure 3.3F). After ≈ 9 days of shaking, large structures were observed (Figure 3.3G).

The structures had 716 ± 11 bonds ($n = 3$, 90% of the 800 bonds in a single decagonal ideal structure), few mismatches 7.6 ± 7(an error rate of 1%), and all eight kinds of vertex star present in a mathematical Penrose tiling.[8] However, the frequency at which each vertex star appeared did not agree with theory; *e.g.* , consider the two vertex stars termed "stars" and "moons" (Figure 3.3G, s and m). Stars and moons are both comprised of 5 fat rhombs but in a different arrangement: tiles in stars are connected by type-a bonds; in moons by type-b bonds. The observed ratio of stars : moons was 19 : 1 (57 stars and 3 moons total, $n = 3$) but the theoretical ratio for a Penrose tiling is $\approx 2.6 : 1$ ($\tau^2 : 1$ by methods in [Sen95]; details are given in Appendix C).

A possible explanation for this discrepancy is that type-a and type-b bonds have different strengths. To check this, fat rhombs were shaken alone (Figure 3.3H) using a similar but shorter annealing schedule (≈ 2 days). The observed ratio stars : moons (now the only possible vertex stars) was $\approx 1 : 5$ (6 stars and 25 moons total, $n = 3$). This reversal in ratio seems inconsistent with the results in Figure 3.3G, but it can be understood by considering unbound edges. For Figure 3.3G the ratio of unbound edges type-a:type-b is $\approx 2.3 : 1$ (173 ± 16, 75 ± 8); for Figure 3.3H this ratio is $\approx 1.3 : 1$ (386 ± 4, 299 ± 8). In both cases minimizing the energy of the system left more unbound edges of type-a; type-b bonds are stronger. I ascribe this asymmetry to the number of wetting changes: type-b edges have two more changes (and hence their menisci more area) than type-a edges have.

[8] An interesting point that I had forgotten until recently is that, apparently, *no other vertex stars form*. This would be trivial if the eight Penrose tiling vertex stars were the only ones allowed by the tile matching rules, but they are not; in fact there are five other vertex stars allowed by the Penrose tile matching rules. See Appendix C for a couple possible explanations.

Figure 3.3: Self-assembly of tiles with increasingly complex wetting codes. Scale bars, 5 cm. (A–D) Checkerboard tiles, 7.5 mm square. (A) Visualization of hydrophilic patches. (B) Disaggregated. (C) Shaken ≈ 30 h. (D) Shifted bonds are strongest at the extremes of subphase density (here 2.4 g/ml) but are present before shaking for all ρ. (E–H) Penrose tiles, 1.2 cm on a side. (E) Two codes (a) and (b) are used. (F) Disaggregated. (G) Shaken $\approx$ 9 days. Note the common "star" vertex star (s) whose "moon" isomer (m) occurs only once, errors ($\star$), trapped tiles (t), and incompletable configurations (inc). (H) Fat rhombs shaken alone. Moons (m) are now common and only one well-formed star (s) occurs. (I–K) XOR tiles, 1.2 cm on a side. (I) Four codes (c), (d), (e) and (f) are used. (J) Disaggregated. (K) Shaken ≈ 60 h. Note errors ($\star$), incompletable configurations (inc), and triangles ($\Delta 1$, $\Delta 2$, $\Delta 3$). (L) A single mismatch may cause a dislocation ($\star$) since mismatched edges are repulsive along 2/3 of their length. Structures in (L) were cropped from a larger image and lightened with Adobe Photoshop.


A
E
a
b
I
c
d
e
f
B
F
J
C
G
t
s
m
*
inc
K
Δ3
inc
*
Δ1
Δ2
D
H
s
m
L
*
*

Independent of any difference in bond strengths one might expect that stars, in the presence of thin rhombs, are more stable than moons. The addition of five thin rhombs to a star yields a structure in which the star is greatly stabilized — four bonds must be broken before the star is destroyed. In contrast, fifteen tiles must be added to a moon before it is similarly stabilized. The stabilization of stars by thin rhombs may be enhanced because thin rhombs bind stars by strong type-b bonds.

XOR tile matching rules require four kinds of nonpalindromic bonds. To enforce them a six-patch wetting code (Figure 3.3I, c-f) was used. The wetting code was designed so that (i) every tile side had three hydrophobic and three hydrophilic patches to cancel the capillary forces' contribution to the position of the tile (ii) every tile side had 3 or 5 changes in wetting character to help make all bonds isoenergetic (iii) if two sides with the correct geometry mismatched, they did so at 4 out of 6 patches, giving the mismatch a net repulsive character. While the codes were nonpalindromic and could be used on square tiles with straight sides, shape complementarity was used to help differentiate the input and output sides of the tiles and prevent shifted bonds.

To show the specificity of the wetting code for XOR tiles, 256 tiles were disaggregated and shaken without an input tile for ≈ 60 h (Figure 3.3K). An average of 362 ± 5.2 bonds ($n = 3$, 75% of the 480 bonds in a square ideal structure) and 13.7 ± 3.8 mismatches formed (an error rate of 3.6%). The four bond types were nearly equally represented ($80 \pm 7 : 88 \pm 1.2 : 99 \pm 7.6 : 95 \pm 3$, c : d : e : f), showing that they were nearly isoenergetic. Errors held in place by a single bond often resulted in dislocations (Figure 3.3L). Without an input tile, the XOR tiles' ideal structure (Figure 3.1J) features dark gray "triangles of

zeros" of all sizes. Experimentally XOR tiles gave complete triangles of size 1, 2, and 3 (Δ1, Δ2, Δ3, Figure 3.3K).

Nucleation of XOR tile assembly on a 4 bit input tile failed; the association of tiles into corners (as in Figure 3.1K) seemed hindered (after 1 day shaking typically 1-2 tiles bound the nucleus). Changing the shape of the tiles, making a flexible input tile, and increasing the rate of shaking in attempts (not shown) to increase the rate of association were unsuccessful. This raised the question: by what mechanism were the large structures observed in unnucleated experiments formed? To answer this, the formation of 275 bonds (present after 2 h shaking without an input tile) was observed by reversing a video of the assembly. Several observations were made. (i) The desired association mechanism occurred infrequently; few bonds (13.5%) resulted from the binding of a tile into a corner in any orientation and fewer still (5.5%) in the orientation required to fit in an input tile. (ii) Most bonds resulted from events involving two multi- tile structures (61.5%). (iii) Dimers were the major active species: 44.7% of all bonds resulted from the association of a dimer to a single tile or multi-tile structure. (iv) The simultaneous formation of two or more parallel bonds (*e.g.* dimerization of dimers to form a square tetramer) was the dominant growth mechanism — 51.6% of all bonds. Thus self-assembly proceeded largely by the assembly of monomers into dimers via single bonds followed by the creation of larger structures via parallel associations.

Measurement of cooperative binding into corners was difficult since tiles bound them so infrequently, but the low error rate (2.8%, similar to longer experiments) and the observed association mechanism together suggest that binding was indeed cooperative. Consider association at a site comprised of two adjacent parallel edges. For such a site there are six

dimers, two correct and four incorrect, that can bind. Without cooperativity to select one of the correct dimers, the error rate would be 1/3, *i.e.* for every two bonds formed, one would expect one mismatch to form. Since 45.8% of all bonds formed by such a mechanism one would expect an error rate $\geq$ 18.6% (expected mismatches $\geq$ 22.9% of bonds, recall %error = mismatches/[mismatches + bonds] = 22.9/122.9). Thus cooperativity appears responsible for an error rate $\approx$ 6-fold lower than expected assuming noncooperative binding.

These experiments show (i) that wetting codes can be used to enforce matching rules for a simple computation and (ii) that the binding of tiles (as dimers or larger species) is reasonably cooperative. The quality of experimental structures followed the trend observed in simulations; checkerboard tiles made the fewest errors (none), Penrose tiles made more, and XOR tiles made the greatest number. These data support the hypothesis that features of a tile set (*e.g.* number of matching rules or whether it is competitive) impact the frequency of defect formation; to draw stronger conclusions future studies must compare the self-assembly of different tile sets using identical wetting codes and annealing schedules. The error rate for XOR tiles was high enough to preclude useful computation but not so high that expected features (triangles of zeros) could not be observed. Finally, quantitative details of bond equilibria and kinetics greatly influenced the structures formed. For Penrose tile structures, the expected vertex stars were present but unequal bond strengths may have contributed to the unexpected vertex star statistics observed. For XOR tiles, an unexpectedly low rate of association into corners was observed; this limits the structures accessible by XOR tile self-assembly to random computations. Finding systems where

self-assembly can be nucleated on an input tile is important for studying the full range of structures suggested by the connection of computation and self-assembly.

3.5 The complexity of growing Penrose aggregates

The self-assembly of Penrose tiles in this chapter takes a back seat to the self-assembly of XOR tiles and the question of whether computational tile sets may be assembled. But the self-assembly of Penrose tilings themselves is quite interesting: at the beginning of this chapter we refer to Penrose tiles as intermediate in complexity between the checkerboard tiles and XOR tiles but there is an important sense in which the Penrose tiles are actually *more complex* than the computational tile sets that we would like to use to simulate Turing machines that can compute arbitrary partial recursive functions!

How can Penrose tiles be more complex than tile sets that simulate Turing machines? In the language of physics, their "correct" assembly requires a "growth rule" that is more complex than that required by our computational tile sets. All of the tile sets that we and others consider to simulate Turing machines require only "local" growth rules for their correct assembly; Penrose tiles require a more complex "nonlocal" growth rule. What does this mean? First, a *growth rule* is really just an algorithm (often nondeterministic) for taking an aggregate with N tiles in it and producing an aggregate with $N + 1$ tiles. Given a particular growth rule, and a set of tiles, the growth rule may or may not produce a desired configuration of tiles. The $\rightarrow_{\mathbf{T}}^{*}$ operation from Chapter 2, incorporated into an appropriate algorithm, forms a growth rule.

A physicist's definition of a *local growth rule*[9] for tile assembly is just a growth rule that decides what tile to add at a site based on a function of the tiles within a constant distance of the site. Put another way, a local growth rule is a growth rule for which the "physics" of the system does not inspect tiles an arbitrary distance away from the site of addition.[10] For the XOR tiles then, a local growth rule for their correct assembly is that tiles binding cooperatively by two bonds are added and tiles binding by one bond are not.

Growth rules, since they are algorithms, may have complexities defined on them. Here we define growth rule complexity as the time it takes for a growth rule to make a single tile addition.[11] If one ascribes an increasing computational cost to the inspection of a tile as a function of distance, either in the space required to represent the position of a distant tile or the time required to propagate information about the identity of a distant tile to the site of addition[12], the concept of growth rule complexity allows a more computational definition for local growth rule: a growth rule is local if it requires no more than a constant amount of computation per tile addition. I will use this definition since it admits nothing as a local growth rule that the physical definition does not and nothing I have ever seen described as a local growth rule violates it. Thus defined, local growth rules appear to capture our idea of

[9]Here we refer to our picture of the "average" physicist, and attempt to distill her philosophy. In fact, the literature of Penrose tilings and quasicrystals is littered with confusing, incompatible definitions for local growth rules and local matching rules. Senechal [Sen95] gives a nice definition for local matching rules under which the Penrose tile matching rules are *local*, even while acknowledging that construction methods for Penrose tilings must be *"global"* in the sense that they must scan arbitrarily large regions of a growing aggregate before adding a tile. What a mess!

[10]More subtly, the algorithm must also select the site for tile addition in a constant amount of time.

[11]I note that this complexity *does not* tell one the time complexity for growing a particular structure. To calculate the time complexity for a structure one needs to know, in addition to the growth rule complexity, other things about the physical world — for example the concentration of tiles. Growth rule complexity talks about how fast a computer can grow a structure, without the limits imposed by diffusion of tiles; growth rule complexity imposes a lower bound on time complexity.

[12]We assume that growth rules, as algorithms, only have access to the same representations of aggregates as physics. For example, the data structure enforces the constraint that information about a change at one edge of an aggregate can only be propagated to the other edge at the speed of light.

physically plausible growth rules[13]. We do not believe that a self-assembly process can do an arbitrary amount of computation at each self-assembly step — unless the computation guiding self-assembly goes on in some hidden universe or extra dimensions (for which we have no evidence), self-assembly (per tile) would have to slow down as aggregates get bigger.

Call an aggregate of Penrose tiles that agrees everywhere with an infinite Penrose tiling a *perfect* Penrose aggregate. Even though a tiling by Penrose tiles is a Penrose tiling if and only if the matching rules are satisfied, one cannot devise a local growth rule based on the matching rules to assemble perfect Penrose aggregates. In fact, no local growth rule exists for the growth of perfect Penrose aggregates [Pen89]; every local growth rule is guaranteed to form incompletable sites[14] like those in Figure 3.1H. To ensure that a Penrose tile is added correctly to a site on a Penrose aggregate, the growth mechanism must inspect tiles at positions arbitrarily far away, an amount of computation that grows as the aggregate grows. Thus, goes an argument that seems correct to me, arbitrarily large *perfect* Penrose aggregates cannot be directly assembled by a physically plausible self-assembly system[15,16]

[13]This idea of "physically plausible" assumes that nonlocal quantum effects play no part in the self-assembly. While my understanding of chemistry and self-assembly does not lead me to include quantum mechanical effects, Roger Penrose does think that the self-assembly of quasicrystals proceeds by some nonlocal quantum mechanism [Pen89, Pen91]. But then he also thinks that quantum gravity is required to explain consciousness and we are pretty sure that neurons can be modeled classically.

[14]The ability to form incompletable sites is not limited to Penrose tiles and is not an accident of their unusual geometry. Any *aperiodic* set of tiles (a set of tiles that admit *only* nonperiodic tilings) have this property [DS95]; thus a simple aperiodic set of 16 square tiles due to Ammann [GS87] can be arranged according to its matching rules to form an incompletable site. Whether all aperiodic tilings fail to have local growth rules is unknown. Interesting though they may be, we avoid aperiodic tile sets for computation since we want physical systems to be able to grow aggregates and compute without errors.

[15]It is an interesting fact and somewhat ironic that we *can* make a self-assembling system that simulates a Turing machine that writes out an arbitrarily long assembly sequence for perfect Penrose aggregates. If we could use this system to guide the assembly of Penrose tiles then, in a sense, self-assembly *can indirectly* make arbitrarily large perfect Penrose aggregates! (Just not in time linear on the size of the aggregates.)

[16]Tilings even more peculiar than Penrose tilings exist. Assuming again the physics is no more than a model of computation, nonrecursive tilings [Han74, Mye74] cannot be created by *any* physical system.

A possible consequence of this deduction, often asserted but incorrect, is that the structure of quasicrystals *cannot* correspond to Penrose tilings and that the basic units that make up quasicrystals *cannot* correspond to units that have Penrose matching rules. Rather, it is possible that the structure of quasicrystals does correspond to *imperfect* Penrose aggregates constructed from units that do have Penrose matching rules. Thus, much of the debate about whether quasicrystals correspond to Penrose tiles has centered on comparisons of the density of defects in experimentally generated quasicrystals with the density of defects in Penrose aggregates generated by local rules (in simulation). More precisely the question is, can local rules for Penrose tile assembly generate Penrose aggregates with a density of defects consistent with experimentally generated quasicrystals (whose assembly mechanism we cannot yet observe directly)? In our experiments, we explored the defect density for Penrose aggregates using tiles that were *engineered* to self-assemble via the Penrose matching rules — thus our experiments seem to directly address this question. Unfortunately, the defect rate that we observed is too high to improve the bound on the defect rate known to be achievable by local rules (theoretically or experimentally) — computer simulations of Penrose tile aggregation with simple local rules have created nearly perfect Penrose aggregates (having just 10 defects) with a hundred thousand tiles [RBJ90]. Thus for our experiments demonstrate a "surprisingly low defect density" for local rules, our experiments would have to have generated nearly perfect Penrose aggregates containing hundreds of thousands or millions of tiles rather than just a few hundred tiles.

Despite their numerical disadvantage, our results with the Penrose tiles have some significance: They represent the first *experimental* demonstration of the assembly of objects

with Penrose tile matching rules and show that a physical system can create aggregates that obey these rules with just a few errors. This is arguably of equal importance to simulations with larger numbers of tiles. And it gives encouragement that a DNA system designed to implement Penrose tiles is worth exploring. Finally, our experiments indicate that physical parameters ignored in simulations so far (for example bond strengths) can have great effect on the distribution of structures formed.

Chapter 4

Error rates and nucleation in DNA lattices

To err is human, and also molecular.

—L. Boltzmann, unpublished letter.[1]

A number of interesting constructions have been proposed for self-assembly, including the Sierpinski triangle (Chapter 1, Figure 1.5, p. 24 and Chapter 3, Figure 3.1L p. 148), the binary counter (Chapter 1, Figure 1.1, p. 12), arbitrary Turing machines (Chapter 2, Figure 2.18, p. 112) and cellular automata [Win98a]; we refer to them later as *computational* constructions. All of these constructions share a special property: At some point during self-assembly, a tile which *can bind* a site on two sides is required to out-compete a tile that *can bind* the site on a single side. If the tile which *can bind* the site on only one side *does* and remains bound, we say that an error has occurred. A further property of our proposed constructions is that the tiles that compose them are consistent with error-free arrangements of tiles that are *different* than the constructions of interest (See Chapter 2, Section 2.6.6). To avoid creating these unwanted structures we must make sure

[1]This was fabricated by me.

that self-assembly starts using specific seed tiles of our choosing; in other words, we must control *nucleation* of our self-assembled structures. Both error-free growth and control of nucleation can be achieved in a self-assembling system for which $\mathcal{T} = 2$. $\mathcal{T} = 2$ conditions are ideal conditions and we may achieve them experimentally only approximately. Thus in any experimental system we must optimize conditions to be as much $\mathcal{T} = 2$ as possible.

Currently, the best candidate experimental system for self-assembly is one based on DNA double-crossover lattices [WLWS98a]. We would like to be able to self-assemble a Sierpinski triangle or binary counter embedded in such a DNA lattice and observe it using atomic force microscopy (AFM). Unfortunately, for the patterns we are interested in making: (i) a few errors create patterns dramatically different from the desired pattern and (ii) since we do not usually get single tile resolution using AFM, the pattern of errors will be difficult or impossible to resolve. Thus it would be very hard to debug an implementation of our constructions.

To this end, we[2] designed a system for studying errors and nucleation in double-crossover lattices which we call the *bar code* lattice. We have begun exploring this system in the laboratory, using as our tools the atomic force microscope and UV spectrophotometer. Our results so far are promising but very preliminary[3].

[2]This chapter represents unpublished work pursued in collaboration with Len Adleman and Erik Winfree.

[3]Thus, most of the experiments in this chapter should be repeated several more times before firm conclusions are drawn.

4.1 Schema for the bar code tiles

We begin with the schema for the first self-assembled 2D DNA lattices, reported in [WLWS98a]. The lattices were constructed using DNA double-crossover (DX) motifs (see Figure 4.3), each bearing four 5-base sticky ends, so that the molecules were geometrically[4] and logically[5] equivalent to the square tiles r and s diagrammed[6] in Figure 4.1a. In the language of Chapter 2, the set of produced assemblies for this tile system (unseeded) at $\mathcal{T} = 2$ would be empty since each tile has only strength-1 bonds. However, for physical models that allow continuous values for $\mathcal{T}$ [Win98a], values of $\mathcal{T}$ less than but close to 2 ($\mathcal{T} \lesssim 2$), the tiles would eventually *spontaneously nucleate* small assemblies via a few strength-1 bonds[7]. These assemblies would grow to yield periodic square-shaped assemblies as seen in Figure 4.1b (see also Figure 4.2a, the caption explains why it isn't square) with very few defects (we show none). The pattern of r and s tiles corresponds to a checkerboard. We have added stripes to the r tiles so that a pattern of stripes emerges; thus we call the pair of tiles r and s the *striped* tiles. The actual DX molecules used to implement the r tiles bear extra DNA hairpins so that under AFM, a similar pattern of stripes is observed (Figure 4.16 and Figure 4.17); we call lattices formed by molecular r and s tiles *striped lattice*.

[4]By this we mean that they are topologically equivalent to squares (*i.e.* they are topologically equivalent to disks) and for all assemblies of DXes, a graph indicating the bonds in an assembly is isomorphic to a subgraph of the square grid.

[5]They had binding domains corresponding to the labels on the tiles in Figure 4.1a.

[6]In this chapter we diagram tiles so that they appear rotated 45°with respect to the tiles presented in Chapter 2; we do this so that tile assemblies may be more conveniently mapped to molecular structures.

[7]See Chapter 2, Section 2.6.5 for a brief discussion of nucleation, as well as Chapter 3 of [Lau70] for a treatment of the kinetics of crystal formation.

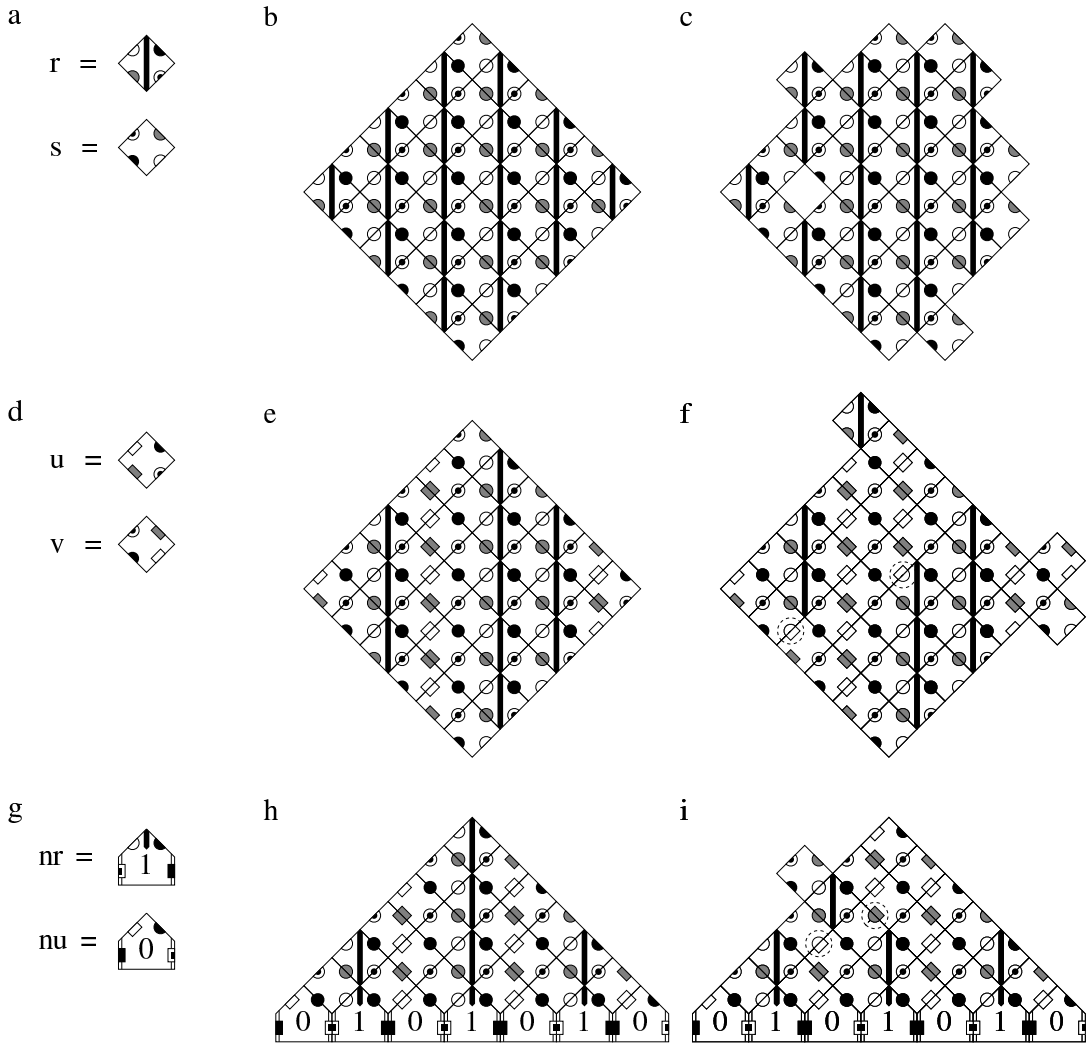


Figure 4.1: Tile system schema for DNA double-crossover (DX) lattices. (a) Tile set for the first DX lattice ever created [WLWS98a] — a *striped lattice* — see Figure 4.16 and Figure 4.17. (b) For $\mathcal{T} \lesssim 2$, periodic square-shaped assemblies should be produced with a few defects (we show none). (c) For $\mathcal{T} \ll 2$ hole defects should appear and the surfaces of the assemblies should roughen. (d) Additional tiles for the bar code lattice. (e) For $\mathcal{T} \lesssim 2$ we expect square-shaped assemblies bearing a random pattern of stripes — a *bar code lattice* — with few defects. (f) For $\mathcal{T} \ll 2$ we expect the same types of defects as in (c) but further we expect errors: places where the binding domains between adjacent tiles mismatch. In the bar code lattice errors show up as changes between striped and non-striped columns. We mark them with dotted circles. (g) Additional nucleating tiles *nr* and *nu* bearing the labels '1' and '0', respectively. (h) Nucleating tiles allow growth at $\mathcal{T} = 2$. Bar code lattice grown on a nucleus of tiles whose sequence is an element of $(01)^*$, should grow without errors at $\mathcal{T} = 2$. (i) Bar code lattice nucleated as in (h) should grow with errors at $\mathcal{T} \ll 2$.

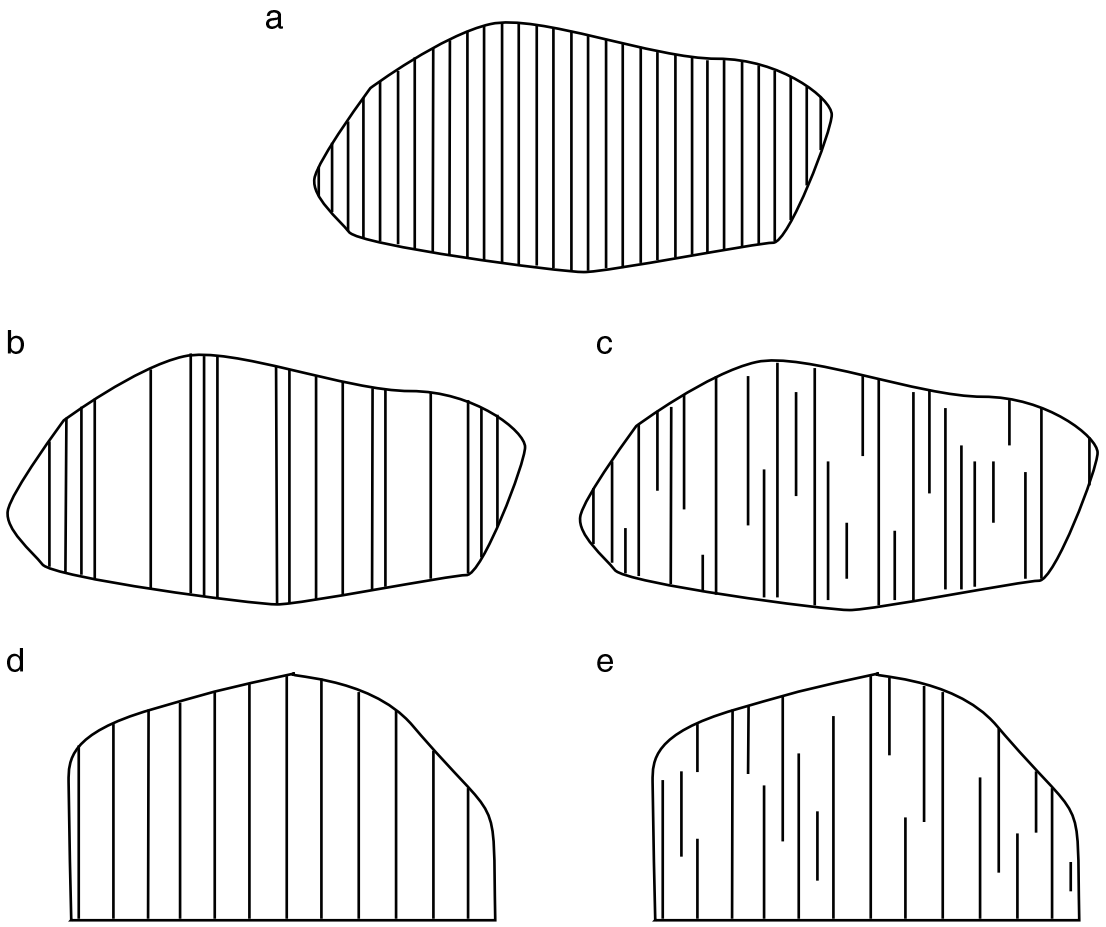


Figure 4.2: Predictions for striped lattice and bar code lattice formation. To clarify the structures we expect for different tile sets under different conditions we give schematic versions of the DNA lattices that we expect to see under the atomic force microscope. (a) Periodic striped lattice (based on Figure 4.1b). We don't expect nice square crystals because the DX units, although geometrically equivalent (see footnote 4) to squares, are not congruent to squares. Based on their shape, if DX units formed perfect crystals we would expect rhomboidal crystals rather than square crystals to form. We don't expect rhomboidal crystals either because the conditions under which the lattices form probably don't approximate $\mathcal{T} = 2$ well enough. Also, even if perfect crystals do form, we are fairly certain that our sample preparation techniques will damage them (see Section 4.5). Compare with Figure 4.16 and Figure 4.17, actual AFM images of DNA tiles corresponding to the stripe tiles. (b) Unnucleated bar code lattice without errors (based on Figure 4.1e). (c) Unnucleated bar code lattice with errors (based on Figure 4.1f). (d) Bar code lattice nucleated on (01)* without errors (based on Figure 4.1h). (e) Bar code lattice nucleated on (01)* with errors (based on Figure 4.1i).

As designed, the r and s tiles do not have the capacity for error that our computational constructions do[8]. That is, regardless of whether an r or s tile binds an assembly by one bond or two, the result is always consistent with the same global striped pattern. Thus, the assembly of striped lattice is not a good model for the computational constructions that we would like to implement in the lab. In principle, for experimental conditions such that $\mathcal{T}$ is significantly less than 2 ($\mathcal{T} \ll 2$), hole defects should appear and the surfaces of the assemblies should become rough with tiles attached by single bonds[9,10] (as well as larger structures grown from them), as indicated in Figure 4.1c. Observe, though, that the basic pattern underlying the assemblies is still the checkerboard pattern.

Since the frequency of such defects should correlate with $\mathcal{T}$, one might be tempted to count them, estimate $\mathcal{T}$, and from this estimate the frequency of errors for our computational constructions. Unfortunately, (i) these features are difficult to observe under the AFM and (ii) even if we had good statistics for hole defects and surface roughness, we

[8]By this we mean that the tiles do not have the *logical* capacity for error, *if* they assemble according to our abstract models of self-assembly. Actually, errors (mismatched binding domains) could occur in a physical system for these tiles if either: (i) A tile sticks to an assembly via a mismatched binding domain. For example, mismatched sticky ends might have DNA sequences that are too similar. (ii) The geometry of a growing assembly departs from the rigid square lattice we envision. For example, a line defect (lattice dislocation) might form — in such cases a row of many mismatches might result.

[9]Note that such defects must occur (at least transiently) in assemblies of striped tiles for *all* temperatures, and that they are not necessarily a bad thing. In order for the r and s tiles to form large crystals, tiles must occasionally bind the face of an assembly by a single bond, otherwise, the assemblies would not grow. For $\mathcal{T} \ll 2$, however, such events become numerous enough to make the faces of an assembly irregular.

[10]To someone familiar with equilibrium crystal growth, our association of the parameter $\mathcal{T}$ with "temperature" will seem very weird or even perverse. In particular, if $\mathcal{T}$ is supposed to have something to do with temperature, our usage seems incorrect because the *best equilibrium shape* for crystals is predicted for low temperature; at high temperature, above what is known as the roughening transition [LNP92] the surfaces of a crystal become irregular and ill-defined. Thus, as far as the equilibrium behavior of crystals is concerned, $\mathcal{T} \sim 1/T$. Somewhat paradoxically, the *kinetic* behavior of crystal growth yields an opposite result; according to a kinetic model at fixed concentration $\mathcal{T} \sim T$. If, for a given concentration of tiles, the temperature is set too low then associations of tiles by single bonds will become "frozen in" and the equilibrium shape of the crystal (no matter how nice it may be) will never be reached. For us, $\mathcal{T}$ is really indicating the dominant mechanism of crystal growth and measures its cooperativity (see Chapter 2, Section 2.2); *$\mathcal{T}$ should not be interpreted strictly as absolute temperature.* In fact, for any *particular* absolute temperature T, we may find tile concentrations and bond strengths that make $\mathcal{T} = 2$.

are uncertain how well our theoretical models would allow us to turn these statistics into quantitative predictions for error frequencies.

Instead, using the tile system for the striped lattice as a starting point, we designed a system in which errors *can* occur and should be readily visualized by AFM. We added two tiles, u and v (Figure 4.1d) which share some binding domains with r and s; together we call the four tiles r, s, u and v the *bar code* tiles. Observe that, by themselves, the tiles u and v could make a lattice with a checkerboard-type pattern; we call the tiles u and v *unstriped* tiles and lattice formed by them *unstriped lattice*. Also notice that:

1. The upper and lower right-hand binding domains of tiles r and u are the same.
2. The upper and lower left-hand binding domains of tiles s and v are the same.
3. The right-hand domains of r and u match the left-hand domains of s and v.
4. The left-hand domains of r and u are different, as are the right hand domains of s and v.
5. The left-hand domains of r match the right-hand domains of s and the left-hand domains of u match the right-hand domains of v.

Let a *column*[11] $\bar{x}$ of tiles refer to a vertically-oriented continuous sequence of tiles labeled x (with at least one tile) that touch at their corners; let $\overline{xy}$ refer to a pair of adjacent columns $\bar{x}$ and $\bar{y}$. From observations 1–5 it can be seen that the bar code tiles can be combined to make a whole family of full square[12] assemblies for which an arbitrary number of $\overline{sr}$

[11] Here a column is the same as a diagonal in Chapter 2.

[12] By this we mean that all sites in an $N \times N$ square are filled with tiles and the binding domains between every adjacent pair of tiles have non-null interactions (following Chapter 2). For the bar code tiles this means their binding domains must match.

can be followed by an arbitrary number of $\overline{vu}$ and vice versa. Letting an adjacent pair of columns $\overline{sr}$ be denoted '1' and an adjacent pair of columns $\overline{vu}$ be denoted '0' then the full square assemblies the bar code tiles can make are essentially characterized by the binary numbers[13] — or equivalently, in the notation of regular expressions, (0+1)*. For $\mathcal{T} \lesssim 2$ we expect that the bar code tiles will make square-shaped assemblies with random patterns of stripes (Figure 4.1e "10110", see also Figure 4.2b) with very few defects — approximating full square assemblies of bar code tiles.

For $\mathcal{T} \ll 2$ we expect errors to form; for the bar code tiles, errors manifest themselves as places where a pair of columns $\overline{vu}$ changes into a pair of columns $\overline{sr}$ or vice versa — these features can be visualized as a place where a stripe begins or ends. Figure 4.1d (also Figure 4.2c) gives an example assembly for $\mathcal{T} \ll 2$ which shows two errors. At the top left face of the assembly a tile has attached by a single bond; the site left and below the tile cannot be filled without forming an error.

Thus, implemented as DX molecules, the bar code tiles should give us an idea of the error rates that we might expect (as a function of experimental conditions) for DNA implementations of our computational constructions. We should be able to find concentrations and temperatures that minimize the frequency of errors. One potential problem is that, in order for bar code lattices to spontaneously nucleate and grow, tiles must occasionally attach via single bonds. In order to get nucleation and growth rates that allow us to observe bar code lattices in reasonable time, we may have to perform experiments under

[13] In fact more than one full square may be associated with the same binary number. For example, both the square depicted in Figure 4.1e, and the same square with its leftmost corner tile u replaced by an r would both be associated with "10110". We aren't trying to *count* the full square assemblies that can be made from bar code tiles so we won't worry about this.

conditions that so favor the formation of single bonds ($\mathcal{T} \ll 2$) that errors will be relatively common.

Given our interpretation of $\overline{sr}$s and $\overline{vu}$s as 1s and 0s, another way to view the unnucleated $\mathcal{T} \lesssim 2$ growth of an assembly of bar code tiles is that of generating and copying a random binary number[14]. The problem of nucleating a bar code assembly to start copying a *particular* binary number, of our choosing, is the same as the problem of nucleating our computational constructions on a particular input string of our choosing. Thus to explore our ability to nucleate computations on a chosen input, we designed *nucleating tiles* (*nr* and *nu* in Figure 4.1g), to add to the bar code tiles, that would constrain them to copy a particular family of binary sequences. Tile *nr*'s top right binding domain matches an *s* tile, and so it should initiate a column pair $\overline{sr}$; tile *nu*'s top right binding domain matches an *v* tile, and so it should initiate a column pair $\overline{vu}$. As marked, the tiles *nr* and *nu* associate by strength-2 bonds[15]. By themselves, one can see that the nucleating tiles can make linear structures that we term *linear nuclei* or when clear, just *nuclei*. Because the nucleating tiles can make arbitrarily large linear nuclei at $\mathcal{T} = 2$, the assemblies do not need to grow by forming occasional single strength-1 bonds. Thus, combined with the bar code tiles in a $\mathcal{T} \approx 2$ physical system, we intend that the nucleating tiles assemble first,

[14]With this point of view, one can see that one interesting mechanism by which unnucleated bar code tiles may make errors depends on a "lack of information". This is because, for an unnucleated bar code lattice to grow in width, new column pairs must be added and hence, new bits of a binary number must be created. Consider the new column pair being entered by the two tiles on the rightmost edge of the assembly in Figure 4.1f; when the two tiles bound, there was no information available as to the identity of the column pair. Having bound, the two tiles disagree on the identity of the column: one is an *s* tile that specifies $\overline{sr}$, the other a *v* tile that specifies $\overline{vu}$. Later, an error will result. Thus the right and left corners of an unnucleated bar code lattice seem like error generators — it appears that 50% of the time, tiles above and below the corner will disagree on the identity of a new column pair. Whether or not such an effect will occur in actual experiments is an open problem.

[15]Actually, the sticky ends used for the DX versions of bar code tiles r, s, u and v are only 5 bases long, while the sticky ends holding the DX versions of tiles *nr* to the tiles *nu* are 32 bases long — an increase in binding energy of much greater than a factor of 2.

via strength-2 bonds, into a linear nucleus before the bar code tiles have time to nucleate their own random assemblies. The bar code tiles should then add to the nucleus via pairs of strength-1 bonds and copy the binary sequence that the nucleus presents — in this case, a sequence satisfying the regular expression[16] $(01)^*$. An example is shown in Figure 4.1h (see also Figure 4.2d). If $\mathcal{T} \ll 2$ we expect to see errors; an example is shown in Figure 4.1i (see also Figure 4.2e).

Thus, the nucleated bar code lattice should be a good model system for studying both errors and nucleation in our computational constructions, in particular for the Sierpinski triangle and binary counter. These last two constructions are very similar to the bar code tiles. In the language of Chapter 2 all three constructions have just four rule tiles; the major difference between them is the organization of their binding domains. Our hope is that if the nucleated bar code lattices can be made to grow with few errors then the Sierpinski triangle and binary counter tile constructions will work under the same experimental conditions with few complications.[17]

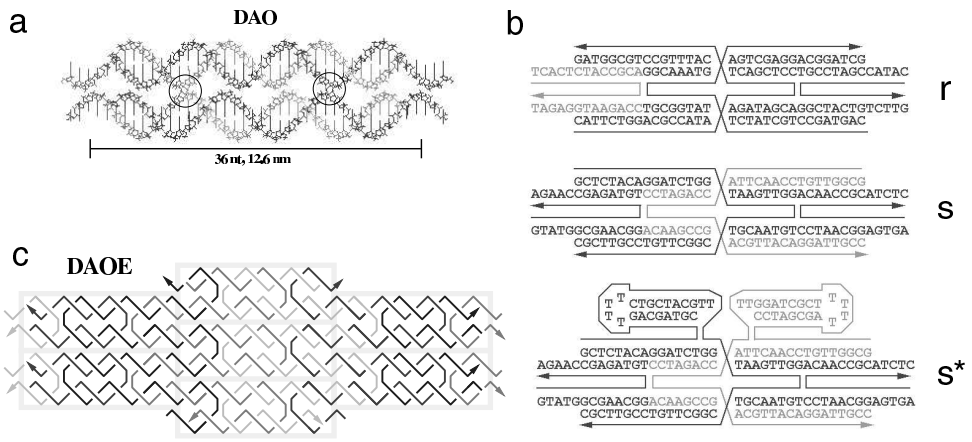


Figure 4.3: Various representations of DX molecules. A figure reproduced from [Win98a] with permission. (a) A molecular model of a DX molecule. (Here a "DAO" molecule [Win98a].) (b) Schematic representations showing the base sequences for strands composing the stripe tiles r and s. s* represents an s tile with extra hairpin loops for contrast under AFM. In our studies we used analogous r tiles with extra hairpin loops (the sequence of the loops was different than that for s*. (c) A schematic showing the topology of strands as they weave through a lattice made of DX molecules.

4.2 AFM results

To begin, we repeated Winfree's experiments with striped lattice. Figure 4.3 shows various representations of DX molecules and the sequences of strands used in our experiments. DNA strands for the tiles r and s were dissolved in 1X TAE, 12.5 mM $MgCl_2$ buffer to a

[16]The reason we express the binary sequence encoded by the nucleus as a regular expression is that regular expressions exactly capture the kinds of linear nuclei that we can make ([Win98a]). Thus, to demonstrate the ability of the bar code tiles to copy nuclei, we'll try other simple regular languages, for example $(011)^*$ or $(01+1)^*$.

[17]A couple potential complications: While all the computational constructions we consider can be made to work with linear nuclei, some of them (in particular the Sierpinski triangle and binary counter) may work better with L-shaped nuclei (for an example of such a nucleus, see details of the binary counter, Chapter 1, p. 12). Another potential complication is that our computational constructions are nonperiodic, often along both lattice dimensions; nonperiodicity may introduce strain into our lattices. (See footnote 15 on p. 10). If we wish to study the effect of 1D nonperiodicity we can study growth of the bar code tiles on nonperiodic linear nuclei; we might study nuclei encoding random strings of 0s and 1s, or we might study a nuclei with a particularly interesting boundary (with respect to strain) namely a long string of 0s next to a long string of 1s. However, the bar code tiles cannot treat the question of whether 2D nonperiodicity may cause too much strain for large lattices to form.

total strand concentration of 2 μM (8 different strands, 4 for each DX tile so that .25 μM r and s tiles were present, respectively). Small volumes (20–25 μL) were placed on freshly cleaved mica disks ($\sim$ 9 mm diameter) attached to small steel disks ($\sim$ .5 mm thick $\times \sim$ 12 mm diameter), and annealed in MJ Research Frame-Seal chambers on glass slides from 90° C to 20° C over time periods ranging from 3 hours to 2 days[18]. As the samples cooled, they passed through a temperature for which $\mathcal{T} \lesssim 2$ and striped lattices spontaneously nucleated. Examples of striped lattice imaged under isopropanol by contact AFM using a Digital Instruments Nanoscope III are shown in Figure 4.16 and Figure 4.17.

Next we obtained[19] DNA corresponding to the bar code tiles. Dissolved to the same total strand concentration as for the striped tiles (2 μM total strands, .125 μM r, s, u, v) and annealed similarly, the DNA bar code tiles spontaneously nucleated to form the structures seen in Figure 4.4 through Figure 4.7. Transitions between striped and unstriped columns (which indicate errors) can be clearly seen in Figure 4.4 and Figure 4.5. Figure 4.6 shows a large field covered uniformly with fragments of bar code lattice; Figure 4.7 shows a close up of this, again showing that striped/unstriped transitions can be resolved. We did not attempt to characterize the error rate very carefully from these images; we hope to obtain larger continuous fragments of lattice for quantitative analysis. However, a couple of things are apparent: (i) The error rate is not so high that distinct striped and unstriped columns cannot be resolved. (ii) The error rate is not so low that we cannot estimate it.

[18]This procedure departs from Winfree's in that annealing took place over the mica surface used for imaging rather than in a plastic PCR tube. We attempted to give the annealing schedules a roughly exponential envelope but the temperature steps were discrete, in 2° C (or sometimes 4° C) increments. Similar results were obtained for different annealing schedules but we have not made a systematic study of this. We note that the absolute temperature for which $\mathcal{T} = 2$ may be different for lattice formation in solution and on mica, thus the absolute temperature at which spontaneous nucleation occurs for our procedure may well be different than for Winfree's procedure.

[19]All DNA used for these experiments was synthesized by Operon Technologies, Alameda, CA or Integrated DNA Technologies, Coralville, IA.

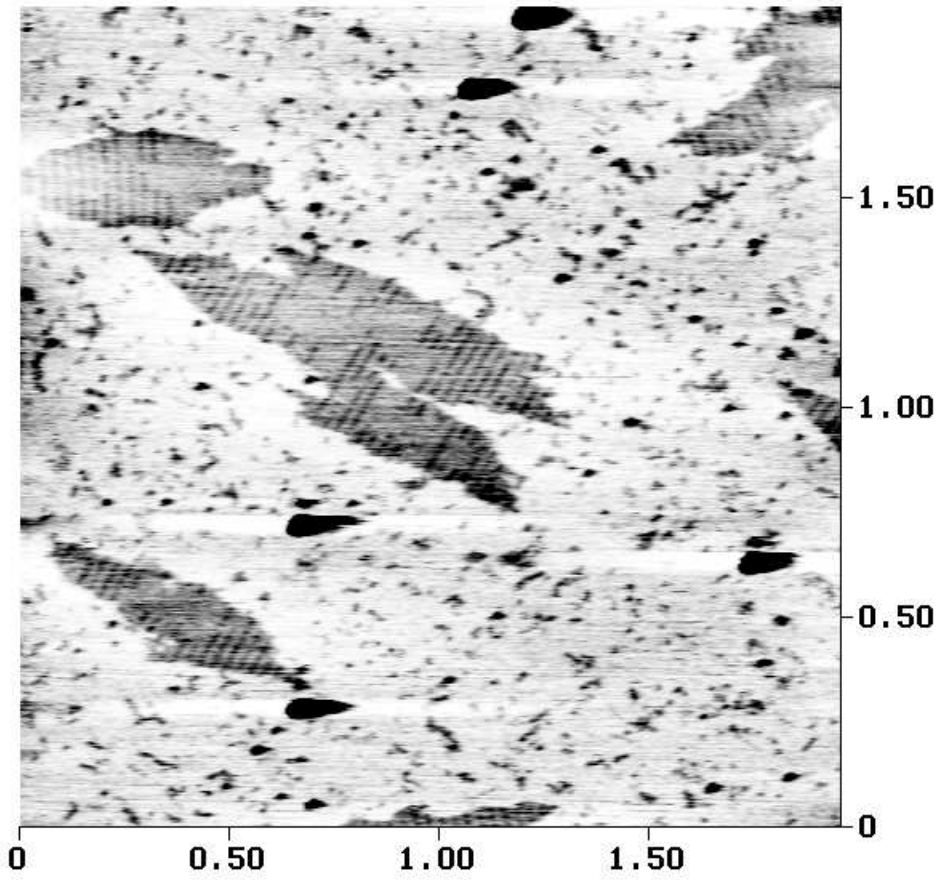


Figure 4.4: AFM image of *bar code* DNA lattices. Axes are labeled in microns. Contrast represents height; the darker a pixel, the greater the height of the surface. DNA lattices have an apparent height of $\sim$ 1–2 nm.

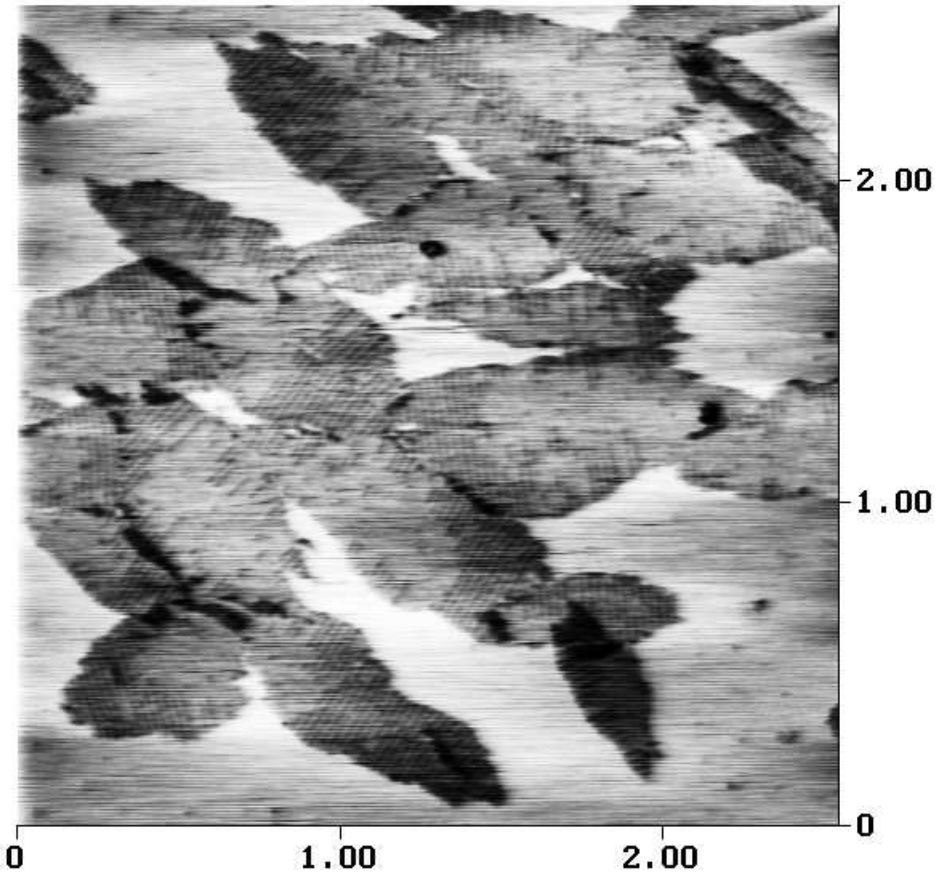


Figure 4.5: Another AFM image of *bar code* DNA lattices. In this image the bar code lattices appear to have patches of striped (rs) lattice and unstriped (uv) lattice which suggests that there is some mechanism for the segregation of striped and unstriped tiles.

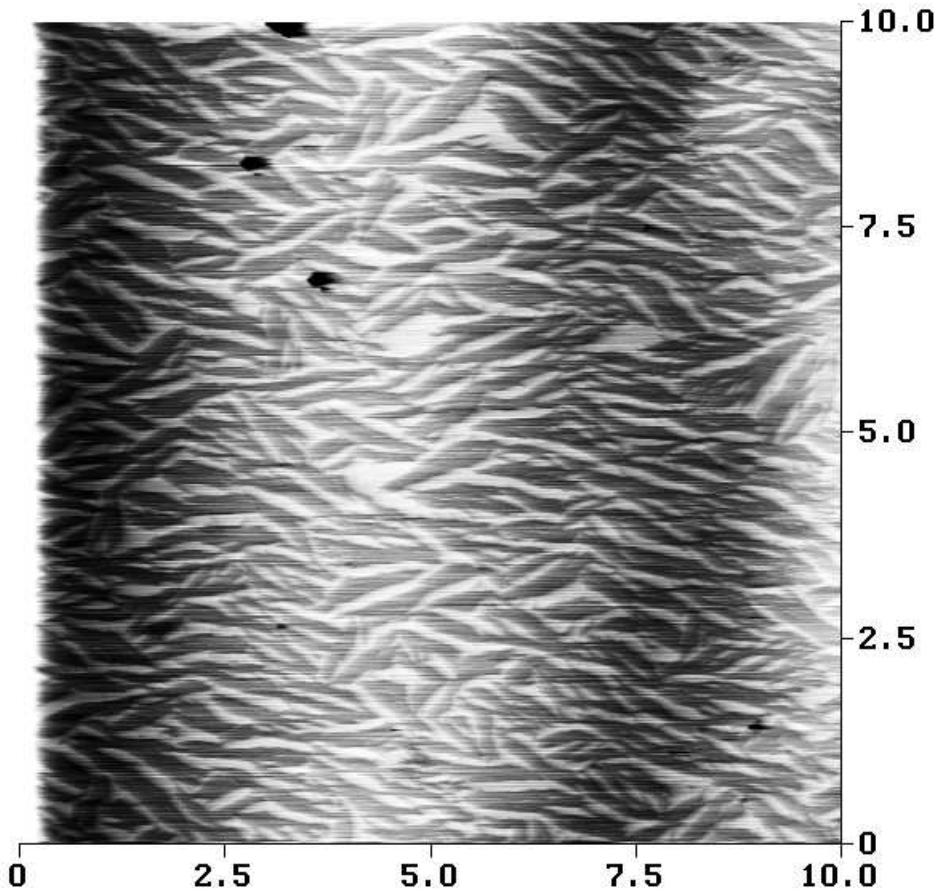


Figure 4.6: High surface coverage of mica by DNA bar code lattice. The small, ($\sim$ .2 micron wide) dark, web-like features are DNA, the light web-like features in the complementary space is the mica surface. The long-wavelength contrast variation ($\sim$ 5 microns) in the image may be actual bowing of the surface, nonlinearity in the piezoelectric scanner, and/or optical interference due to a poorly adjusted laser.

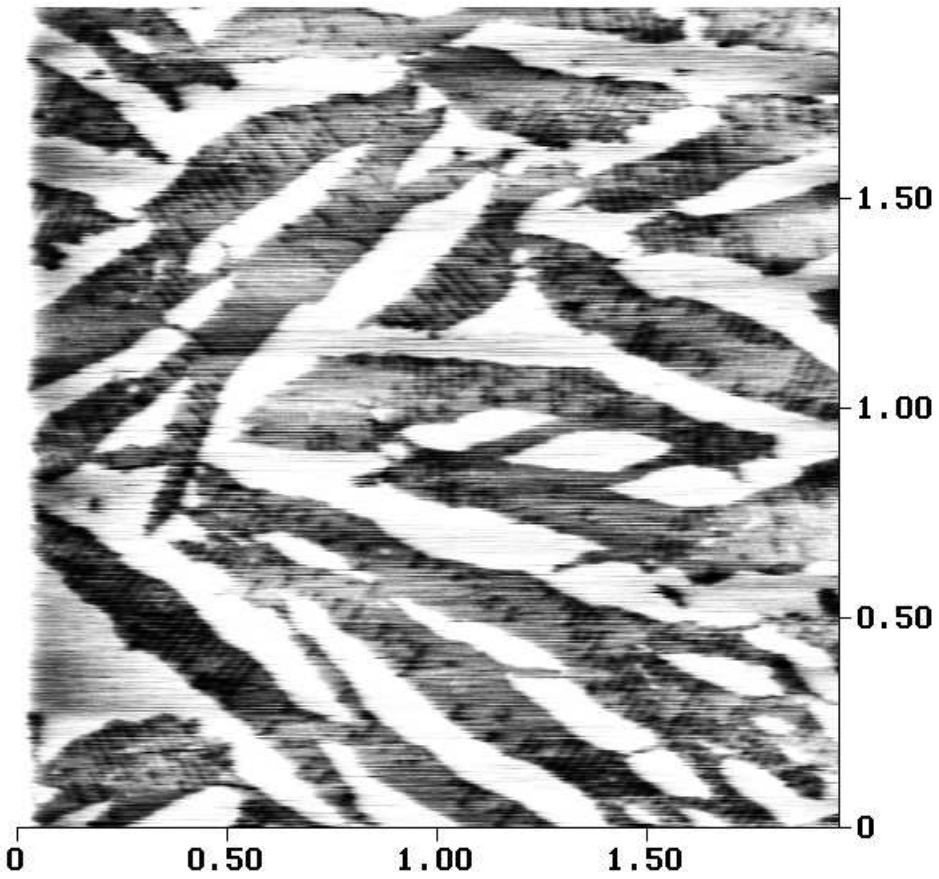


Figure 4.7: A close-up of the sample from Figure 4.6.

Observe that the width of the lattice fragment in Figure 4.4 is approximately 200 nm. Thus there are approximately 100 double helices across its width; DX molecules are each 2 helices wide so in a particular column of tiles, then there are roughly 50 DX tiles. There is an opportunity for error every time a tile in a particular column (say an r column) is added[20], and when a tile in its matching column (say an s column) is added. Thus there are 100 opportunities for error in each column pair across the width of the lattice. Very roughly, it looks like there is an average of 1 or 2 errors across the width of the lattice for each possible striped/unstriped column pair. Thus we would guess that the error rate is roughly 1–2% — that is to say, 1–2% of the time a tile is added, an error is formed.

Figure 4.5 shows something unexpected; by eye, it appears that the bar code tiles sometimes segregate into patches of striped and unstriped lattice. This is *not* predicted by the binding domains present on the tiles. Looking across the width of a bar code lattice there should be no correlation between adjacent pairs of columns; an $\overline{sr}$ should be followed by an $\overline{sr}$ with probability 50% and followed by an $\overline{vu}$ with probability 50%. We have not measured the correlation and compared it to a random distribution but the images are suggestive. Perhaps there is a small strain energy associated with putting striped tiles next to unstriped tiles, or perhaps the DNA hairpins on the striped tiles have a slightly favorable interaction that cause them to segregate. Whatever the reason, it would seem to complicate an attempt to measure error rates in unnucleated lattices and pose a problem for algorithmic self-assembly.

[20] We assume that when bar code tiles are added, they bind either by their top two bonds or their bottom two bonds as they would if growing from an already existing nucleus.

Our hope then is that, by nucleating lattices with the tiles *nr* and *nu*, we will be able to force the bar code tiles to copy a particular pattern of stripes, namely a pattern from the set (01)*. In doing so we believe that we will avoid some of the difficulties inherent in measuring error rates for unnucleated bar code lattice outlined in the last section, and perhaps sidestep[21] the issue of segregation of striped and unstriped lattice. We have done preliminary experiments in this direction by annealing DNA bar code tiles (1.$\bar{3}$ μM total, .08$\bar{3}$ μM for each bar code tile — close to the concentration used for Figure 4.4) with DNA nucleating tiles at different stoichiometric ratios (1:1 bar code:nucleating, .08$\bar{3}$ μM for each nucleating tile; also 10:1, .008$\bar{3}$ μM; 100:1, .0008$\bar{3}$ μM; and 1000:1, .00008$\bar{3}$ μM). At stoichiometric ratios of 1000:1 and 100:1, we observed fragments of unnucleated bar code lattice and no hint of nucleated lattices or linear nuclei. At 10:1 we saw very small fragments of lattice and/or very small linear structures; it is difficult to judge. We seemed to observe larger unnucleated fragments at higher stoichiometric ratios but this observation needs to be repeated[22]. At a stoichiometric ratio of 1:1, we observed filamentous structures like those seen in Figure 4.8a and b. Given the lengths of the sticky ends (32 bases) on the nucleating tiles it is surprising that we do did not see long linear filaments at a 10:1 ratio of bar code to nucleating tiles.

[21] We do not mean that the problem of segregation of striped and unstriped lattices will go away — the energetics that drive the segregation will still be present, and may increase the probability of errors. However, we *will* be measuring the error rate in a system that most closely matches the nucleated schemes in which we are most interested (the Sierpinski tiles and counter tiles). Thus, whether it helps or hurts, we will not have to guess how segregation changes the picture for nucleated lattice growth.

[22] Each experiment was performed in duplicate — out of eight samples, three (one sample each at 1:1, 10:1, and 100:1) yielded no interpretable DNA structures having either large areas of flat mica or areas of what we believe to be DNA and salt debris. This irreproducibility is a common problem for our AFM experiments. Clearly sample preparation needs to be perfected — see Section 4.5. A further complication is that past experiments by Erik Winfree suggest that the surface adhesion of linear DNA structures is highly dependent on concentration of the linear structures.

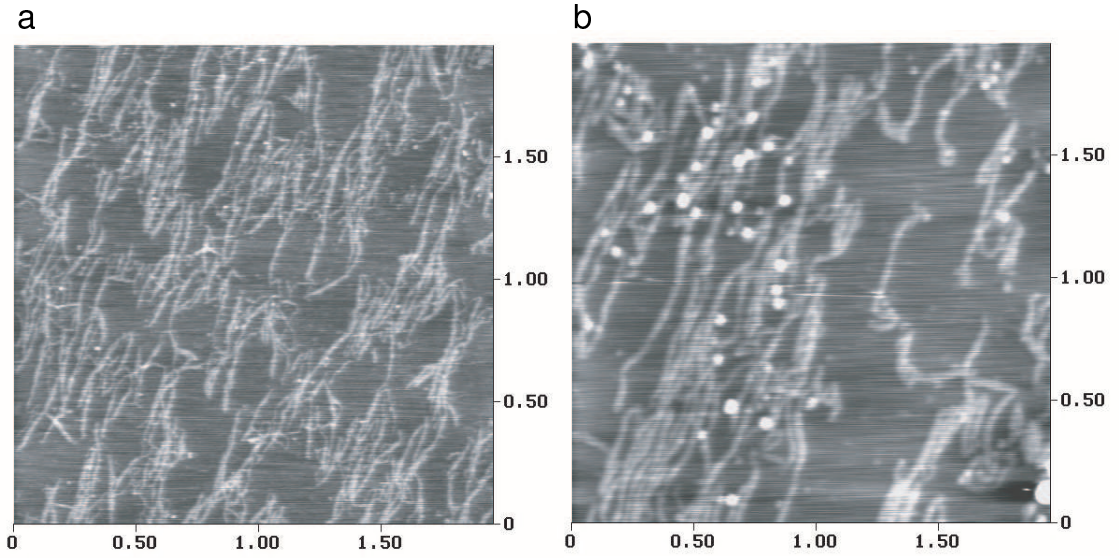


Figure 4.8: AFM images of *nucleating* DNA tiles with bar code tiles at a 1:1 ratio. Contrast again represents height but here the lighter a pixel, the greater the height. DNA filaments have about the same apparent height as lattices (1–2 nm). (a) Filaments form. Dots on filaments hint that we might be seeing the beginnings of stripes. (b) Up close, it appears that many of the dots may be small salt crystals.

One possible explanation for these results is that, for the annealing schedule that we used, the association of the nucleating tiles to form linear nuclei and the association of bar code tiles to form lattices do not really occur independently (as we would like). These events are supposed to be isolated by temperature — nucleating tiles, even at very low concentration, are supposed to associate at a temperature much higher than that at which the bar code tiles can begin to associate with the nucleating tiles or themselves. But, perhaps in the presence of a high concentration of bar code tiles, the nucleating tiles don't find each other, but rather associate with bar code tiles. Since, on their bottoms, nucleating tiles cannot associate with other bar code tiles they would act as defects, possibly terminating growth of the bar code lattice. This is consistent with the observation that while we saw no clear evidence of linear nuclei at stoichiometries of 10:1, 100:1 and 1000:1, the size of the bar code fragments that were observed got larger — there were fewer nucleating tiles around to terminate their growth. Perhaps, at high enough relative concentration, the nucleating tiles had an opportunity to find each other rather than bar code tiles and formed the filaments we observed at 1:1 stoichiometry. If this hypothesis is true, and our problem is that the spontaneous nucleation of bar code tiles are competing with the formation of long nuclei (whether or not nucleating tiles are terminating bar code lattice growth), we may be able to make long linear nuclei in a separate test tube, by themselves, and introduce them to the bar code tiles at a temperature well below the melting temperature of the nucleating tiles interactions[23]. Alternatively we could make covalently linked nuclei, or try a different annealing schedule. Whatever our method of generating long nuclei,

[23]One can imagine a procedure by which a small amount of bar code tiles are added continuously to pre-formed nuclei so that the concentration of bar code tiles is always optimal (that is, they associate with the nuclei but their concentration is too low for them to spontaneously nucleate lattices on their own).

it is important that the eventual stoichiometry of bar code tiles:nucleating tiles be high. Otherwise, our nucleated bar code lattices will be too narrow, in terms of the number of rows of bar code tiles, for the error rate to be analyzed.

Despite the difficulties noted, our AFM results are promising for two reasons: (i) we can observe the striped/unstriped transitions that we believe are associated with errors in unnucleated bar code lattices and (ii) we can affect bar code tile assembly by adding nucleating tiles to create filamentous structures similar to the linear nuclei that we would like to create.

4.3 DNA melting studies

The preliminary AFM studies described above made it clear that we need an experimental method that allows us to: (i) quickly test different experimental conditions and find parameters that allow us to control nucleation and (ii) understand the functional relationships between experimental conditions and phenomena such as supersaturation, nucleation time and the size of critical nuclei. An ability to observe the growth of lattices in real time would allow us to extract these parameters and relationships quickly, but real-time temperature-controlled AFM of DNA lattices is currently beyond our capabilities. In the meantime, observing the assembly of DNA lattices by UV absorbance is one easy way to try to get at this information, albeit indirectly. In this section we give evidence that UV absorbance allows us to observe DNA tiles in a supersaturated state and that we can get some idea of their nucleation time.

The UV absorbance of single-stranded DNA at 260 nm is much greater than the absorbance of double-stranded DNA at 260 nm. Thus, we can use a UV spectrophotometer

to distinguish between double- and single-stranded DNA — per nucleotide, single-stranded DNA is "darker" at UV wavelengths. In particular, we can use this difference to follow the *melting* of a pair of complementary DNA strands from their hybridized state to their free state as the temperature is increased. Thus, we plot a *melting curve* showing absorbance versus temperature. Because DNA melting is a cooperative process, the melting curve that can be obtained for a simple pair of complementary DNA oligos is a sigmoidal curve. Typically we also anneal (cool) the strands and watch to see if they re-associate according to the same curve. If the DNA strands are at equilibrium at every point during the heating and cooling process, the curves will be exactly the same[24]; for short oligos melted at 2° C/min nearly identical curves are usually obtained. Note that for simple complementary oligos (at micromolar concentrations) without unusual secondary structure, the *slowest* one would usually heat and cool normal oligos is .1° C/min; at this rate no hysteresis between heating and cooling curves should be observed. This will be important later since DNA tiles that form lattices are often not at equilibrium — not when cooled at such gentle rates, nor even when held at constant temperature for over a day!

The first melting curve[25] we present is for a pair of non-interacting DNA tiles, in particular, the tiles s and v. Figure 4.9a shows the melting curve for all 8 strands of the

[24]In order to get such matched curves, volume loss to evaporation must be prevented — otherwise the absorbance of a sample will slowly drift upward with time, especially at high temperature. Reluctant as we are to mess up our nice DNA solutions or goop up our nice quartz cuvettes, we have learned that adding mineral oil on top of our samples to prevent evaporation yields melting curves that are much more reproducible. Unfortunately, the melts in this chapter were done without mineral oil and some of the hysteresis observed is probably due to evaporation. We did perform the melts in tightly capped cuvettes with a small air space (~ 100 μL) above the sample but this causes other problems — the vapor pressure above the sample increases with temperature. We had one cuvette crack under the stress. Another practical note: the first melt of a solution often yields poor data because as the solution heats up for the first time, it degasses. The resulting bubbles pass through the light path, scatter light, and add occasional big spikes to the absorbance signal.

[25]All melting curves were performed using a Varian Cary 300 UV/VIS spectrophotometer.

tiles s and v, dissolved to a total concentration of 2 μM (.25 μM each strand, .25 μM each tile), from 20° C to 90° C (black curve) and back to 20°C (gray curve), at a rate of 2° C/min.[26] Figure 4.9b is the first derivative of the cooling trace in Figure 4.9a[27]; peaks in this plot correspond to inflection points in Figure 4.9a. There are a couple things to notice. First, rather than a simple sigmoid, the curve appears to have two inflection points. This indicates that there were two different melting events occurring, one at ~53°C and the other at ~63°C. We believe that this should be attributed to the formation, at different temperatures, of different domains in a single DX molecule (rather than be interpreted as very different melting temperatures for each of the two tiles s and v) since melts of single tiles have given similarly complex melting curves. Not all DX molecules exhibit this effect to the same degree so we believe that this effect is highly sequence specific. Second, the cooling curve does not return to exactly the absorbance value measured at the beginning of the heating curve. This discrepancy may occur because of evaporation or because the sample was cooled very quickly; we believe the curves would match better at lower heating/cooling rates (especially at low temperature where reaction rates are low).

More interesting is the case in which we heat and cool DNA tiles that can form a lattice. For them we should be able to watch at least two transitions: (i) a small sigmoid at low temperature corresponding to the hybridization of sticky ends between tiles and (ii) a large sigmoid at high temperature corresponding to the hybridization of the component strands of the tiles into complete tiles. One question that arises is whether the small

[26] These curves were taken after the strands had already been heated to 90° C and cooled to 20° C.

[27] Computed using the Cary WinUV software package using a filter of width 9. We take as the melting temperature the peak of the derivative plot. It is intended to indicate when half of the molecules are hybridized and half are not; this is an approximation, however and holds to the extent that DNA hybridization can be modeled as a two state process.

sigmoid should be observable; the sticky ends are quite small and their UV absorbance hybridization signal might be lost in the large signal that results from the formation of the tiles. We note that for DNA tiles without hairpins[28], the ΔOD[29] of the large transition should be about 6.4 times as large as the ΔOD of the small transition since there are 64 base pairs internal to each tile and 10 base pairs involved in sticky ends per tile.

Figure 4.9c shows a melting curve for the striped tiles r and s at a concentration of .25 μM each tile. Not shown is melting data for the *first* heating and cooling cycle. Upon first heating we observed that there was a *single*, large ΔOD, high temperature transition at about $\sim$63°C. The first cooling curve is similar to the second cooling curve, the black curve in Figure 4.9c, which we describe later. Upon heating a second time, the absorbance signal becomes noisy (the gray curve in Figure 4.9c). The noise appears to increase in magnitude until about 40°C, after which it appears to decrease in magnitude until about 50°C, disappearing to leave a smooth curve at approximately 60°C. Upon cooling at 2° C/min (the black curve), as expected, there is a high temperature transition at $\sim$63°C (indicating formation of the tiles r and s) and a second, small ΔOD, low temperature transition at $\sim$27°C (indicating association of the short sticky ends between the tiles). The high temperature transition is slightly sharper upon cooling than the high temperature transition observed upon initial heating. The ratio of ΔODs for the two transitions observed on cooling is approximately[30] that predicted: $\frac{\Delta \mathrm{OD}_{high}}{\Delta \mathrm{OD}_{low}} = \frac{.86-.695}{.695-.67} = \frac{.166}{.025} = 6.64$.

[28] All melts in this section were performed using tiles without the extra hairpin loops that give contrast under AFM. This was done so that the melting curves for the different tiles r,s,u, and v would be as similar as possible and so that the small signal for the sticky ends would be as large as possible compared to the large signal due to tile formation.

[29] That is, the change in *optical density* of the solution observed at 260 nm.

[30] We chose where one transition ended and the other began by eye, in an unprincipled manner. We need to redo this calculation in a principled way, fitting the appropriate sigmoids to the curves. The important

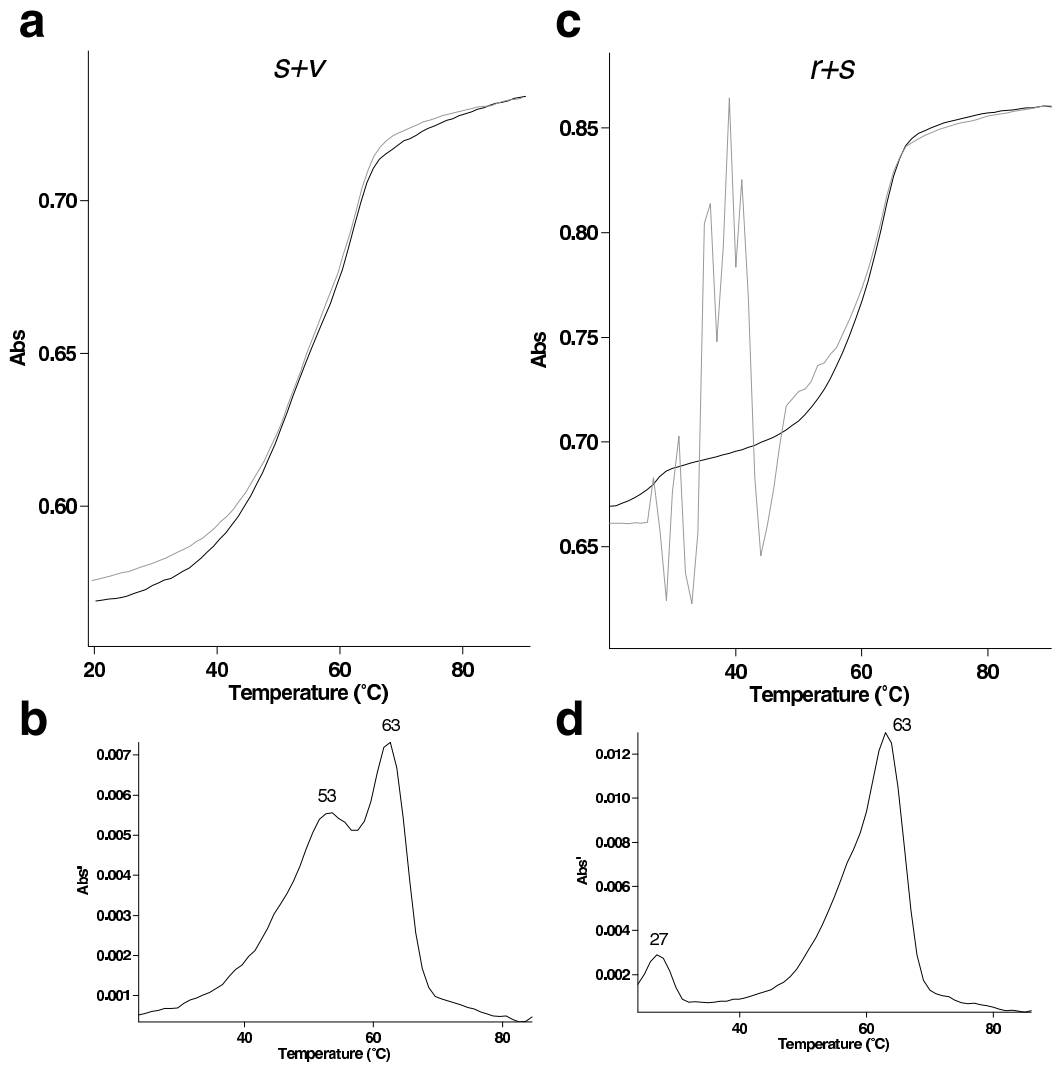


Figure 4.9: Melts of DNA. All samples were heated to 90° C and cooled to 20° C once before the curves shown here were taken; thus these data are representative for second and subsequent heatings and coolings. (a) A melt of the unrelated bar code tiles s and v. The tiles were heated from 20°C to 90°C (gray curve) and then cooled back to 20°C (black curve). (b) The first derivative of the cooling curve in (a). Peaks are meant to indicate melting temperatures, for which half of the strands in a particular hybridization reaction have associated. All temperatures written next to peaks are approximate. (c) A melt of the lattice forming tiles r and s. (In order to make the absorbance of r and s tiles equal, a version of r without extra hairpins was used.) A large scattering signal is observed below 50°C and is not completely abolished until 60°C (d) The first derivative of the cooling curve in (c).

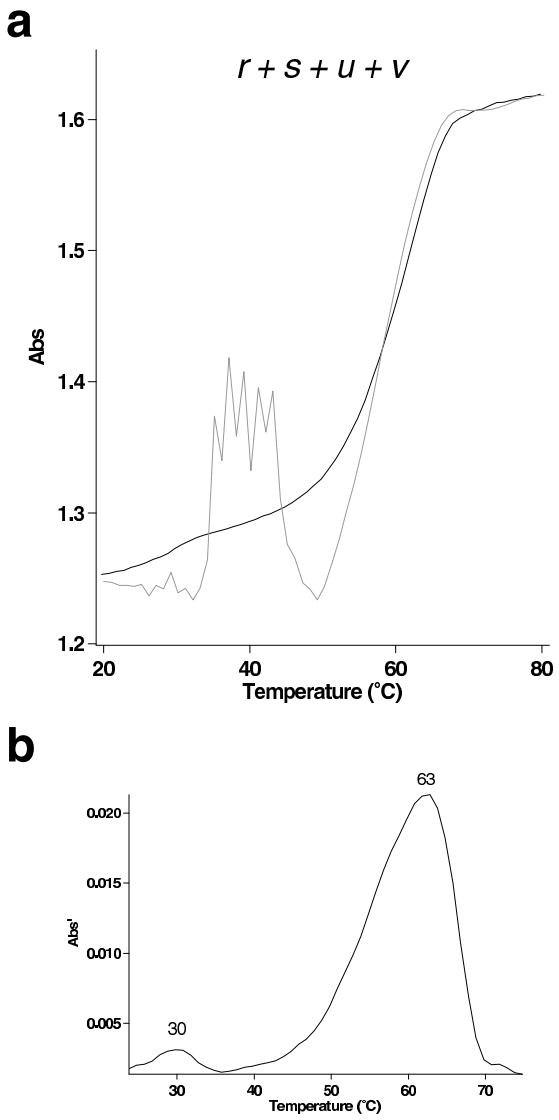


Figure 4.10: Melt of the bar code tiles. (a) A second melt of the bar code tiles r, s, u, and v (representative of the second and subsequent heatings and coolings). (b) The first derivative of the cooling curve from (a).

Figure 4.10 shows second melt data for the 16 strands of the bar code tiles at a total concentration of 4 μM (.25 μM each tile). It is very similar to that for the r and s tiles. The individual concentration of any tile is the same as for the previous lattice melt — thus we observe the same high temperature transition at about 63°C. The presence of 4 different DX molecules with slightly different melting curves gives this transition a broad but smooth shape. For the bar code tiles (in comparison to the striped tiles) the concentration of sticky ends to which a matching tile (having the complementary sticky end) may bind has, for certain sticky ends, been doubled[31]; this is probably why the low temperature transition is observed at a slightly higher temperature — 30°C. The noise signal observed upon a second heating, is very strong. The noise signal, although not repeatable in its actual values, appears to be a real and reproducible effect — subsequent heatings give noise signals over a similar range of temperatures, depending on experimental conditions. We discuss this later.

The first question we ask is: why for the stripe tiles (or bar code tiles) is there only one discernable transition upon the initial heating (rather than small transition or noise), and why might it appear broader than that observed upon cooling? We believe that when individual strands for the bar code tiles are mixed at room temperature[32], without ever being heated to higher temperature, the strands for the DNA bar code tiles are not at equilibrium: there is a kinetic barrier for the formation of a tile from its component

point is that, although the low temperature transition is mediated by short sticky ends, it is still easily observable at the DNA concentrations used.

[31] In particular, this is true for the sticky ends representing the right sides of the tiles r and u and the sticky ends representing the left hand sides of the tiles s and v

[32] In this chapter, by room temperature we mean approximately 20° C.

strands since the strands must wind around each other to form a complicated pseudo-knotted structure. Our picture is that at first, at room temperature, the component strands are all involved in a loose tangle of partially formed tiles whose sticky ends do not hybridize particularly well, especially since they are unaided by the stacking interactions available to them when complete tiles associate. From a comparison of the absorbance before first heating to that after the first cooling we estimate that >95% of the base pairs are hybridized in this tangled state. This diverse mixture of base pairing interactions, from a few hybridized sticky ends to many half-tiles to a few complete tiles, melt out over a broad range of temperatures. Hence, on a first heating, the melting transition is broad. This picture is supported by gel experiments in which DNA strands for a single DNA tile were mixed and left at room temperature for a period of three hours, and an identical tube was heated to 90° C and cooled to room temperature. A non-denaturing gel (data not shown) indicated that the annealed strands had almost completely reacted to form whole tiles, while the room temperature tube had formed very few complete tiles (unquantitated). Thus the initial melting of the striped tiles or bar code tiles is our first example of how the melting behavior of lattice forming tiles departs from that of a normal equilibrium melt[33].

The second question we ask is: why are the heating and cooling curves so different? Also, will the features of the curves (lots of noise and a two phase melt) be reproduced upon subsequent heating and cooling cycles? It turns out that what happens on a particular heating cycle depends exquisitely on how fast the previous cooling cycle was and how long

[33]We note that it must be the case that the tile s and v are not at equilibrium before the first time they have been heated and cooled, but because they do not form lattices, the difference up subsequent heatings and coolings is not so striking.

the solution rested at room temperature. If the previous cooling cycle was fast (2° C/min) and the subsequent re-heating initiated immediately then, upon re-heating, the two phase curve was very nearly reproduced. If, however, the cool down was slow or the sample rested many minutes at room temperature, the curve observed in the next heating cycle is vastly different — it exhibits a great amount of noise in the absorbance signal (see the gray line in Figure 4.9c for the striped tiles and Figure 4.10b for the bar code tiles). Although we do not have sufficient data to make quantitative statements, the magnitude of the noise observed and the range of temperatures over which it was observed also seemed to correlate with the rate of cool down and the length of time at room temperature. The slower the cooling and the greater the inter-cycle time, the greater the magnitude of the noise and the broader the range of temperatures over which it was observed. If cooling was fast and the inter-cycle time small, the noise might have a magnitude of only .2 OD and occur only from 35° C to 40° C. We note also that the low temperature transition was also observed under these circumstances. If cooling was slow and the inter-cycle time long, the noise could be on the order of ±.5 OD (on the order of total absorbance signal for striped tiles or bar code tiles) and could start at 20° C and persist through 60° C. Under this condition, no low temperature transition can be observed, even if it exists, because it is hidden under the noise signal. Clearly the bar code tiles are *not* at equilibrium in these experiments. Paradoxically, the most reproducible melting curves (with the least noise) were obtained at the *fastest* heating and cooling rates with the *smallest* inter-cycle interval times! This makes no sense because we are giving the DNA less time to equilibrate. What is going on?

Upon first seeing these data, we were at a loss to understand the noise. Then John Petruska walked by and said, "Ha! What you are seeing is the Tyndall effect — light scattering off of particles in solution." What we were seeing was the scattering of our 260 nm light from crystals of DNA bar code lattice, when the crystals had grown to about the same size as the wavelength of the light. If we had UV vision, we could look at our solutions of DNA tiles at room temperature, and see that they would look milky because of the DNA crystals in them! It is interesting to note that many of the crystal domains in our AFM experiments (*e.g.* Figure 4.4 through Figure 4.7) are about 200–300 nm in size — just big enough to start scattering strongly. Thus we believe that the noise signal, from now on *scattering* signal, is diagnostic for the formation of DNA lattices in solution[34].

Our explanation of the melt data then, is this: Upon cooling the tiles quickly to around 40° C (the temperature we guess to be the melting temperature[35] for the crystal), the tiles become supersaturated; since we do not linger, few lattices form. Upon cooling quickly the rest of the way to room temperature, the tiles condense into loose networks based primarily on single sticky end interactions rather than forming large crystals. Perhaps the structures are very small crystals rather than networks; we are unsure. Whatever the structure, most of the sticky ends are hybridized by 20° C. The attendant drop in absorbance, from about 35° C to 20° C, is the low temperature transition that we observe.

[34]We do not fully understand the nature of the scattering signal that we observe. Naively, one would expect that light scattering should only *increase* the apparent absorbance of a sample, by scattering light away from the detector. However, the absorbance signal observed due to light scattering during melting (or very slow cooling) falls well below that expected from a quick cooling curve. Thus it seems that scattering actually increases the amount of light reaching the detector. This phenomenon, known as *forward scattering* is actually well understood although we do not yet understand the theory.

[35]The melting temperature for a crystal is the temperature such that: (i) above the melting temperature a continuous distribution of crystalline sizes is predicted at equilibrium with small crystals being the most common (ii) below the melting temperature a distribution of crystals is predicted but a single large crystal (a solid phase) is also predicted.

Upon fast re-heating many of these structures are melted and we observe the transition in reverse through about 35° C (only if the re-heating is immediate and very fast). However, the bar code tiles are *still below* their melting temperature and thus some of the structures may grow to critical size, start to grow large lattices *even though we are heating the sample*, and scatter light! Their growth may, in fact, be aided by the heating since the melting of other small crystals is providing them with monomer and the diffusion rate is increasing. Finally, as about 40° C is passed, the crystals begin to melt, and the scattering signal begins to decrease. The reason that we see a larger scattering signal over a broader range of temperatures when we cool more slowly and increase the inter-cycle time is that we are giving the lattices more time below their melting temperature to nucleate and grow into large lattices.

The data most strongly supporting this point of view is that *if the cooling cycle is slow enough, then* below 40° C *the sample begins to scatter during the cooling cycle and the scattering gets larger and larger with time (and as the sample cools)!* Figure 4.11 show the results of an experiment in which bar code tiles were heated to 80°C and cooled to 40°C quickly (2°C/min). Then the bar code tiles were cooled slowly (.5°C/min) to either 36°C (Figure 4.11)a or 37°C (Figure 4.11)b and held at constant temperature for at least 2000 minutes (33 hours). Strikingly, the absorbance does not change much for a characteristic length of time (3 repetitions of each experiment yielded very similar results); then drops of exponentially for a time; then becomes hidden in a light scattering signal. Thus it appears that, initially, the bar code tiles are in a supersaturated state. This state persists until nuclei of a critical size have formed and can grow to a size that they begin to scatter light. We call the length of time the solution persists metastably in the supersaturated

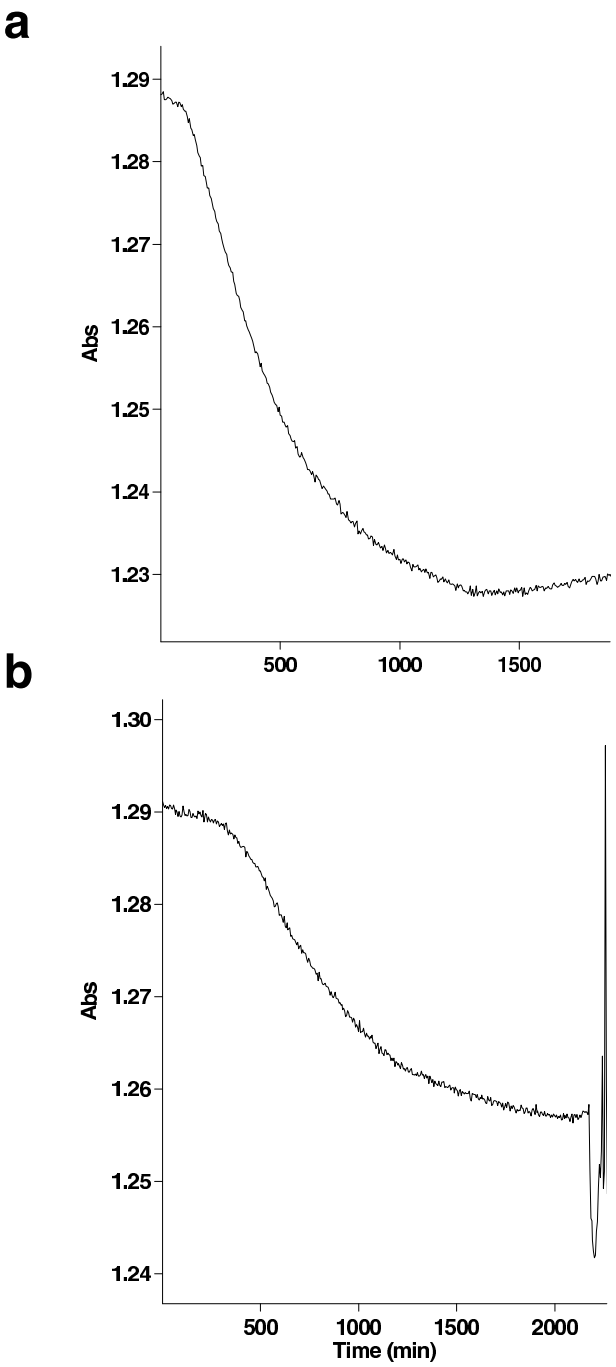


Figure 4.11: Bar code tiles can be prepared in a supersaturated state. (a) At 36°C lattices do not nucleate and grow for $\sim$ 120 minutes ($\sim$ 2 hours). We attribute the rise in absorbance after $\sim$ 1250 minutes ($\sim$ 21 hours) to light scattering. (b) At 37°C lattices do not nucleate and grow for $\sim$ 360 minutes ($\sim$ 6 hours). Strong light scattering occurs after about 2200 minutes ($\sim$ 37 hours). Note that the time scales in (a) and (b) are slightly different.

state the *nucleation time*. The nucleation time at the melting temperature for the bar code tiles should be infinite. It is interesting to note that it is certainly very sensitive to temperature — a change of 1°C changes the nucleation time by a factor of three from about 2 hours at 36°C to 6 hours at 37°C. We note also that the absorbance signal in these experiments is still changing after 36 hours at constant temperature — this indicates just how far away the bar code tiles are from equilibrium when we cool them at 2°C/min or even .1°C/min!

Next we consider the nucleating tiles and their effect on the bar code tiles. Figure 4.12a shows melting curves for the nucleating tile *nr* at a total strand concentration of 1 μM (.25 μM each tile).[36] The *nr* tile appears to have a major transition at 67°C, although, like other DX molecules may have another lower temperature transition ($\sim$ 58°C). Figure 4.12b shows melting curves for the combined tiles *nr* and *nu* at a total strand concentration of 2μM. The 67°C transition for the nucleating tile *nr* (and presumably *nu*) is still observed. An even higher temperature transition, at 76°C is observed. We believe that this transition is the DNA strands for the 32 base long sticky ends of the nucleating tiles associating *first*, at a higher temperature than the melting temperature of the nucleating tile molecules themselves. This was not a conscious design choice but we do not believe that it should cause any difficulties.

What effect do nucleating tiles have on bar code tile assembly? In our hands, so far, small concentrations of nucleating tiles have little effect. Figure 4.13 shows the melting curves that result when 4μM bar code tiles were mixed with 2μM nucleating tiles. The

[36]The absolute absorbance measured in these experiments is greater than that observed for the other DX DNA tiles. This is because the nucleating tiles have very long 32 base sticky ends that add a larger background absorbance.

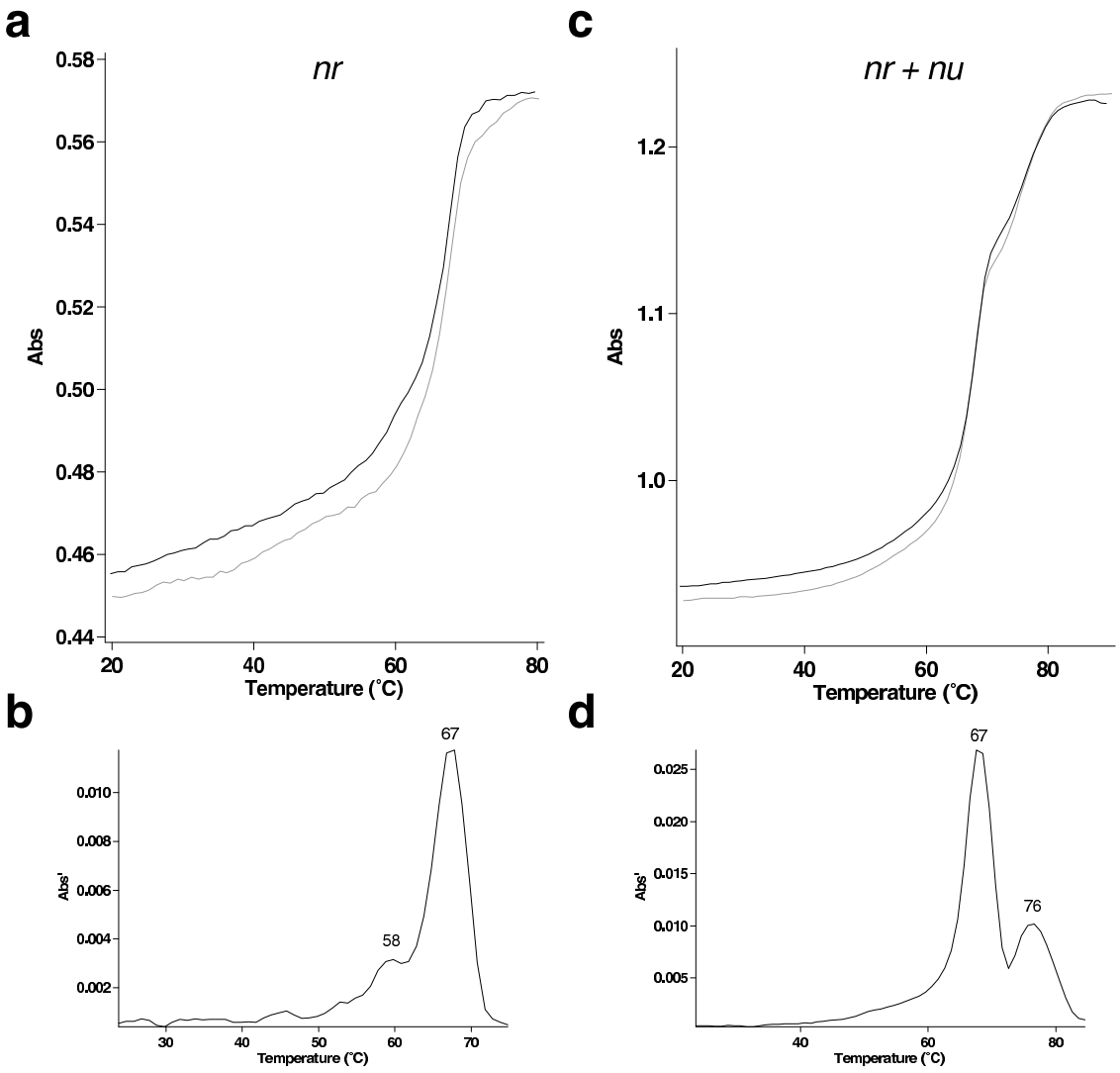


Figure 4.12: Melts of the nucleating tiles. (a) A melt (second) of the tile *nr*. (b) First derivative of the cooling curve in (a). *nr* has a slightly higher melting temperature (67° C) than the tiles *r*,*s*,*u*, and *v*. (c) A melt (second) of the pair of tiles *nr* and *nu*. (d) First derivative of the cooling curve in (c). The hybridization reaction for the sticky end interaction between tiles *nr* and *nu* has an observed melting temperature of 76° C. Since this is higher than the melting temperature of the individual nucleating tiles, these sticky ends actually associate first, upon cooling, before the tiles nucleating tiles have formed.

tiles r, s, u, v, nr, and nu are thus present in equimolar amounts (.25 μM each tile). Upon heating, no light scattering is observed, and the curve has a low temperature transition that closely matches that observed in the cooling curve. Repetitions[37] of this experiment suggest that such a large concentration of nucleating tiles added to the bar code tiles abolishes light scattering no matter how slow the cooling nor how long the inter-cycle time. Thus we believe that the nucleating tiles are preventing crystals of light scattering size from forming. This is consistent with our AFM studies that show that a stoichiometric amount of nucleating tiles restrict the structures that are formed to long filamentous structures.[38] We now need to perform studies of the effects of nucleating tiles on nucleation times for the bar code lattice. Presumably, the presence of long linear nuclei formed by nucleating tiles should significantly reduce the nucleating time since bar code tiles should be able to assemble on the nuclei as soon as they encounter it. We have seen that this is the case for stoichiometric concentrations of nucleating tiles. However, small concentrations of nucleating tiles, in a proportion that would yield large nucleated bar code lattices are necessarily present in low concentrations. Thus the absolute rate at which nucleated lattices use up tiles is limited. If nucleated lattices grow too slowly, spontaneous nucleation will occur and the population of crystals may be dominated by undesirable spontaneously nucleated crystals. So far, results with small concentrations of nucleating tiles have proven inconclusive.

[37] In one experiment we cooled from high temperature to 40°C as normal and then stopped for 30 minutes every 2°C until room temperature was reached. No light scattering was observed but in the control experiment with just bar code tiles, strong light scattering began below 30°C.

[38] This is also further evidence that the light scattering signal is diagnostic for the formation of large DNA lattices. We believe that light scattering in a UV melt should be used to assay the formation of lattices using new DNA motifs since it is quicker, easier, and more consistent than AFM experiments. Given a new lattice forming motif that *does form* lattices, one could perform a few unlucky sample preparations that leave no lattices on the mica and incorrectly declare that the motif is a loser prematurely.

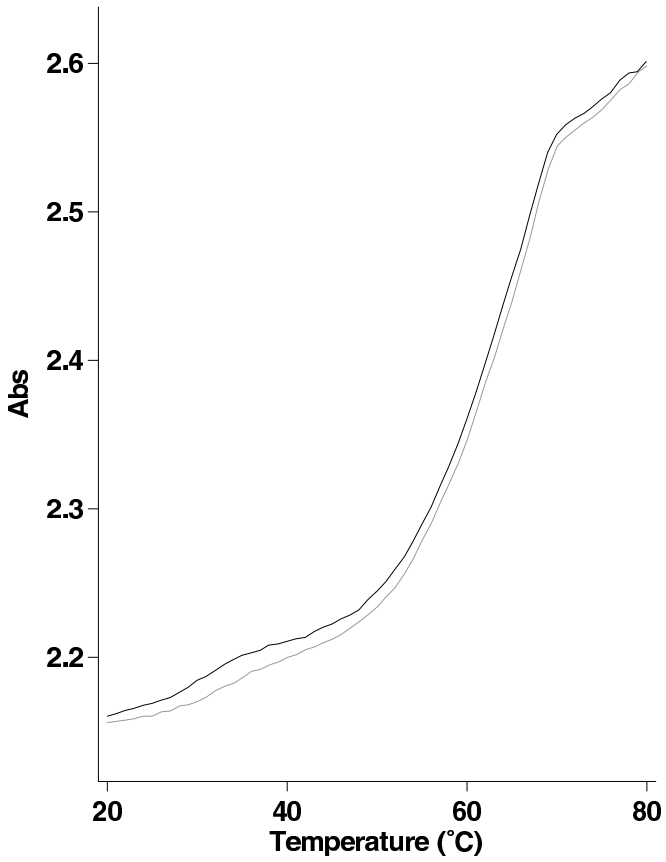


Figure 4.13: An equimolar concentration of nucleating tiles suppresses the crystallization of bar code tiles. Here the concentration of all the tiles r, s, u, v, nr, and nu is equimolar (.25 μM) and no light scattering is observed upon heating.

Our initial melting studies are promising for several reasons: (i) we can observe both a high temperature transition associated with tile formation and a low temperature transition associated with tile-tile association; (ii) we appear to have a diagnostic signal (light scattering)for lattice formation[39]; (iii) we can observe supersaturation of the bar code tiles and achieve nucleation times on the order of 6 hours — in this window we hope to be able to achieve good growth on our own artificial nuclei; (iv) at high concentrations nucleating tiles abolish the light scattering signal, clearly affecting the formation of bar code lattices.

4.4 A theoretical difficulty.

Our scheme for observing large nucleated bar code lattices hinges on our ability to make long linear nuclei from the DNA versions of the nr and nu tiles. How long, in principle, may the nuclei grow? According to an equilibrium analysis of sticky end hybridization between nucleating tiles, very long indeed[40]. The nucleating tiles have two complementary pairs of sticky ends. The pair of sticky ends that mediate the interaction $nr \cdot nu$ are denoted $nr\cdot$ and $\cdot nu$ when free. This is the pair of sticky ends we analyze; the other pair can be treated similarly. The sticky ends are themselves very long, 32 bases, so that an estimated equilibrium constant for their association[41] is quite large: 3.75×10^{35}/M for 20° C and 3.87×10^{23}/M for 40° C. If we assume that we start with a 1 μM concentration of each

[39] Although this also means that we cannot measure UV absorbance when DNA lattices are present.

[40] In this section we ignore that the nucleating tiles are themselves held together by domains smaller than the sticky ends mediating the interactions between nucleating tiles.

[41] We calculated the equilibrium constant as $K = \exp \frac{-\Delta G^\circ_s}{RT}$, where $\Delta G^\circ_s = \Delta H^\circ_s - T\Delta S^\circ_s$, $\Delta H^\circ_s = -8s$ kcal/mol, and $\Delta S^\circ_s = -22s - 6$ cal/mol/K. For $s = 32$ at 20° C, $\Delta G^\circ_s = -48$ kcal/mol and at 40° C, $\Delta G^\circ_s = -34$ kcal/mol.

nucleating tile then at equilibrium[42], at either temperature, the effective concentration of hybridized ends $[nr \cdot nu]$ is essentially 1 μM; the concentration of free ends $[nr\cdot]$ or $[\cdot nu]$ is 1.6×10^{-15} μM at 20° C and 1.6×10^{-9} μM at 40° C. Assuming for a moment that all of the $nu \cdot nr$ interactions were made, these concentrations would imply that at 40° C the average nuclei is 625 million $nr \cdot nu$ units (or 1.2 billion tiles) long and at 20° C the average nuclei is 6×10^{14} $nr \cdot nu$ units (or 1.2×10^{15} tiles) long![43] Remembering that the $nu \cdot nr$ interactions will also be at the same equilibrium halves these numbers. If we assume that we have 1 ml of solution with 1 μM of each tile, this means that at 40° C there are 2 million nuclei, averaging over half a billion tiles long; at 20° C we expect to have just *two* nuclei averaging 6×10^{14} tiles long. If we decrease the concentration of nucleating tiles 1000 fold, to 1 nM, then at 40° C there will be 60,000 nuclei, averaging 20 million tiles long. Even if we decrease the concentration of nucleating tiles to 1 pM, then at 40° C there will be 2000 nuclei averaging 600,000 tiles long. Thus, even between 35° C and 40° C where we plan to hold the bar code tiles in the supersaturated state, equilibrium analysis predicts that a small concentration of nucleating tiles should generate long nuclei.

[42]Concentrations reported solve the equation

$$K = \frac{[nr \cdot nu]}{[nr\cdot][\cdot nu]}$$

[43]While equilibrium theory predicts that at 20° C all the tiles are essentially in just two long molecules, kinetic theory predicts that such monstrous molecules do not form very quickly — certainly not on time scales that we can observe in lab. As the tiles stick together, their concentration drops, and the *rate* of the association reaction drops quickly — as the square of the remaining concentration of free tiles. Consider just the last step, for which four nuclei merge into just two nuclei. The effective concentration of a sticky end (of any type) at the four nuclei stage is 2 ends/ml or 3.3×10^{-21}M and the forward rate constant for short DNA oligos (we assume that the sticky ends diffuse as short oligos, even though they are really attached to long high molecular weight nuclei) is 6×10^5/M/sec. The forward rate is thus no faster than $6 \times 10^5[3.3 \times 10^{-21}]^2$ M/sec $= 6.5 \times 10^{-36}$ M/sec. We only need a change from to 3.3×10^{-21}M to 1.6×10^{-21}M (1 sticky end/ml of each type) but this will take 2.5×10^{14} second, or about 8 million years! Thus, while at equilibrium there may be two giant molecules, we do not believe that kinetically such a state is ever reached.

We told Bernie Yurke our plans for making linear nuclei for the bar code tiles by combining stoichiometric quantities of tiles *nr* and *nu*. He said, "Aren't you worried about the 'stoichiometry' problem?" We said, "What stoichiometry problem?" Bernie explained that making long linear self-assembled structures doesn't just depend on the equilibrium constant for association between tiles but that it also depends on the stoichiometry of the tiles being very close to 1:1.

Here, we give a logical explanation, using a very unphysical model of self-assembly, that gets to the point. Consider a world such that when we put a finite number of nucleating tiles *nr* and *nu* into a fixed volume, they reach a state for which the number of bonds between them is maximum. From our analysis above, this seems like a good approximation of equilibrium at room temperature (20° C) for the nucleating tiles. It also seems that for an equal number x of *nu* and *nr* tiles this is the best outcome we could hope for; it yields linear nuclei of length $2x$. As a concrete example consider mixing the 10 tiles in Figure 4.14a — they would assemble to a single 10-tile long product Figure 4.14b. Now consider mixing tiles with unequal stoichiometry, eight copies of the tile *nr* with five copies of the tile *nu* as shown in Figure 4.14c. Figure 4.14d–f show states of the system for which the number of bonds is maximal. Figure 4.14d is a state that we find acceptable; an 11-tile long nucleus is formed. However, there are many other isoenergetic states, for example Figure 4.14e and f, for which the average length of a nucleus is the same as for Figure 4.14d, 13/3, but the length of the longest nuclei formed is much smaller. We note that these states are also roughly isoentropic[44] with the state shown in Figure 4.14d; because all three states are roughly isoenergetic and isoentropic the system should spend roughly the same amount

[44]Having the same number of particles, perhaps not the same number of conformational states.

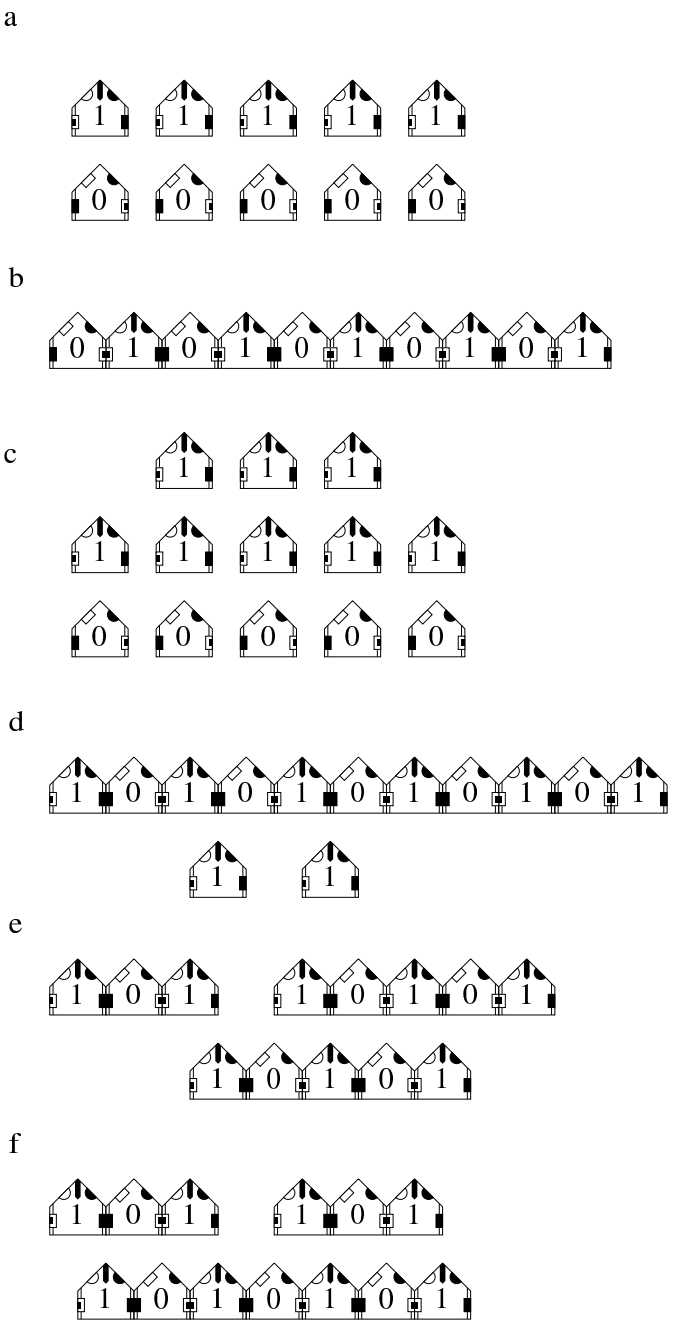


Figure 4.14: Possible effects of stoichiometry on nucleating tile assembly. (a) We imagine mixing five copies of the tile *nr* and five copies of the tile *nu*. (b) Assuming a physical system that makes all possible bonds, a single 10-tile long nuclei results. (c) Now we imagine eight copies of the tile *nr* are mixed with five copies of the tile *nu*. (d) One possible set of structures compatible with maximizing the number of bonds includes an 11-tile long nuclei. (e),(f) Other possible sets of structures maximize the number of bonds but give much shorter nuclei.

of time visiting all three states. Thus, without counting the number of such states, we guess that a typical state for the system has a distribution of nuclei lengths grouped much more tightly around the average than is the case for our desired state, Figure 4.14d.[45]

How bad might this effect be in general? Notice how extra tiles affect the states available to the nucleating tiles by starting with the state in Figure 4.14b and adding extra tiles nr. Ignore the first extra tile that goes on the end to form an 11-tile nucleus; then each additional tile nr is capable of breaking the long nucleus at one position. Now consider that we have a concentration c M of nucleating tiles nu and a concentration $c+x$ M of nr tiles. Assume that the sticky ends are long enough and the temperature low enough so that essentially all bonds that can be made are made. Then the number of bonds that are made is proportional to $2c$ but the maximum number of breaks is proportional to x and the average length of nuclei could be as low as $2c/x$.

Assume that the maximal number of breaks allowed by the stoichiometric excess are made; then the length of nuclei will be fairly sensitive to the stoichiometry between the tiles nr and nu. If the ratio is 1.01:1 we expect our nuclei to be, on average, 200 tiles wide. Each nucleating tile is approximately 25 nm wide (twice the width of a bar code tile) so that the width of a nuclei with 200 tiles would be approximately 5000 nm or 5 microns. At $\mathcal{T} \approx 2$ such nuclei could grow into a triangle 100 tiles high or about 400 nm high (each bar code tile is 4 nm wide). This should be easily visibly under the atomic force microscope,

[45]Why do we wish to have the distribution of lengths skewed towards a single long nuclei? One reason is that we wish to maximize the amount of lattice — the total number of tiles that have bonded to nuclei. The size of a lattice formed by a nucleus of length l at $\mathcal{T} = 2$ is $\sim l^2$ tiles. Consider a distribution of k nuclei of lengths $x_1 \ldots x_k$ with the constraint that the number of nucleating tiles is fixed, that is, $\sum_{i=1}^{k} x_i = C$ and for all i, $x_i \geq 1$. If we assume that each nuclei makes a full-sized lattice (the number of bar code tiles is not limiting) then the total amount of lattice is $L \sim \sum_{i=1}^{k} x_i^2$. For a fixed k, L will be maximized when there are $k-1$ nuclei of length 1 and a single nuclei of length $C-k+1$. We omit the proof here but it is easy to see that rearranging two nuclei of lengths $x_i > 1$ and length $x_j > 1$ into a pair of nuclei of length $x_i + x_j - 1$ and length 1 always yields a greater amount of lattice.

and we might hope to get reasonable error statistics from such a lattice. However, we believe that our uncertainty in the concentration of the nucleating tiles is at least 10%[46]. Thus the ratio of *nr*:*nu* may well be 1.1:1 and we should expect nuclei on average 20 tiles wide — only 500 nm wide, yielding bar code triangles only 10 tiles or 40 nm in height. We would not expect to get good error statistics out of such small lattices. Even worse, this analysis seems conservative; it treats bar code tiles as atomic objects. In reality each nucleating tile is itself a DX molecule composed of 4 DNA strands — deviations from 1:1:1:1 for their stoichiometry should cause similar breaks in the nuclei and further decrease the average length length of a nucleus.

Our conclusion is that we really need to make sure that long nuclei are forming. This will involve AFM and non-denaturing gels of just the nucleating tiles, without any bar code tiles present, to measure the length of the nuclei that form. If Bernie's objection is indeed experimentally a large problem, and the nuclei are very small, we have several choices. We can try to adjust the stoichiometry of the nucleating tiles so that it is more even. Within a single nucleating tile we can make the stoichiometry very close to 1:1:1:1 for its constituent DNA molecules by mixing up the DNA strands for a nucleating tile at concentrations that are close to correct and running the result on a non-denaturing polyacrylamide gel; correctly formed nucleating tiles with all four strands will migrate as a single band — extra strands and malformed complexes will migrate as different bands. Thus a correctly formed nucleating tile with exactly 1:1:1:1 stoichiometry can be recovered

[46]We quantitate DNA concentrations by UV absorbance, given that we have calculated the absorbance/M/cm of a particular sequence using a computer program. Different programs for predicting the UV absorbance yield results that vary by at least 5%. This would not be a problem if the difference were completely systematic but there appear to be sequence dependent differences of a few percent. After adding a few percent pipetting error when working with small volumes and our uncertainty is probably more than 10%.

by cutting out the appropriate band. Getting the stoichiometry of whole tiles $nr : nu$ close to 1:1 is more difficult. One approach is to conduct a search for the ratio of nu and nr tile stock solutions that yields the longest nuclei. This can be accomplished by mixing up a range of solutions at various concentrations and observing, by non-denaturing gel, which ratio yields the highest molecular weight nuclei.

Another way to make long covalent nuclei is to generate them via PCR. We have had good experience making long repetitive DNA sequences well in excess of 20 kilobases in length via the method of William Stemmer [SCH+95]. Unfortunately, single continuous DNA strands are incompatible with the particular DX motif (DAO) we have been using for making the bar code tiles. DX molecules and the lattices they form can be thought of as tiles and partial tilings, but in reality they have much more complicated structure. If one imagines all of the nicks in a DX lattice closed by ligation, the lattice can be thought of as a fabric, with strands of DNA winding back and forth, up and down, in the fabric. In lattices formed by DAO DX molecules, no strands traverse the fabric, either vertically or horizontally, through a single row or column of DX molecules that could act as a nucleus. Instead, as shown in Figure 4.3c, strands wind back and forth between *two vertical columns*. There exist DX motifs with the right topology (DAE) for incorporating single continuous strands of DNA as nuclei, but the DNA strands for all of our bar code tiles would have to be redesigned and re-synthesized. See [Win98a] for a schematics of the different DNA lattices that can be made with different DX motifs.

4.5 Sample Preparation

A major stumbling block to a quantitative understanding of the formation of DNA lattices is the irreproducibility of the experimental procedure for producing lattices. The original procedure used by Winfree [WLWS98a][47] goes as follows:

1. Approximately 50 μL of buffer containing the DNA strands for the desired lattice are annealed in a PCR tube.

2. About 10 μL of buffer are transferred by pipette to a freshly cleaved mica surface affixed to a stainless steel disk. The solution is allowed to stand for $\sim$ 2 minutes.

3. Excess salts are removed by pipetting an additional 100-200 μL of de-ionized water onto the mica surface, allowing it to sit for $\sim$ 1 minute, and shaking the droplet off.

4. The sample is then dried by gently blowing a stream of "canned air" (Dust-off) across the mica surface at a low angle.

5. The mica surface is flooded with isopropanol.

6. The sample is imaged by contact mode AFM.

This method of preparing and imaging DNA lattices often works very well. It enabled proof of the structure of DNA lattices and gives a favorable lower bound on how large they can grow — in [WLWS98a] a single domain DNA lattice is reported that is 2μm$\times 8\mu$m and contains approximately 300,000 DX tiles. Yet this procedure has a few drawbacks. First, the surface coverage of the mica by DNA lattices is often sparse and uneven. The largest

[47] Suggested to Winfree by Bob Moision at San Diego State University.

field of view that may be scanned using the AFM scanner we currently use is approximately $10\mu m \times 10\mu m$. It is often the case that dozens of these fields must be scanned before any DNA lattice is found; this can be very frustrating to the AFM operator. Second, there is little quantitative reproducibility in surface coverage or size of lattices observed. In what are a couple dozen repetitions of Winfree's procedure, we have never observed a DNA lattice as large (or even half as large) as the $2\mu m \times 8\mu m$ lattice reported. Third, the quality of lattice observed varies: sometimes the lattice appears to be a "tight weave" and is continuous; at other times it appears to be a "loose weave" and bears numerous small holes and fractures.

We attribute the irreproducibility of lattice density, size, and quality to mechanical damage[48] caused by the sample preparation procedure. First, we believe that pipetting the DNA from solution to the surface (step 2) causes some shredding of the lattices. It is well know that passing double stranded DNA through syringe tips breaks megabase long genomic DNA into fragments about 15 kilobases long [KB91] — until researchers quit passing DNA through syringe tips they thought all DNA was less than 15 kilobases in length. Assuming that pipetting fragments DNA lattices to a similar size we should never observe DNA lattices greater than about 5 microns on a side. Second, and perhaps more importantly, we believe that capillary forces induced by dewetting the surface (step 4) rip DNA off of the surface in a highly irreproducible manner[49], sometimes leaving large numbers of DNA lattices, sometimes leaving a few, sometimes stretching the lattices into

[48] The irreproducibility could be attributed to other factors, for example, the quality of the mica surface — perhaps some mica cleaves are "stickier" for DNA than others. I discount this hypothesis because of the consistency of the results I achieved when I imaged samples in tapping mode under buffer rather than perform steps 3–6 above.

[49] This is because the drying of the surface is a manual step; it is difficult to aim the air stream and control its flow rate consistently by hand.

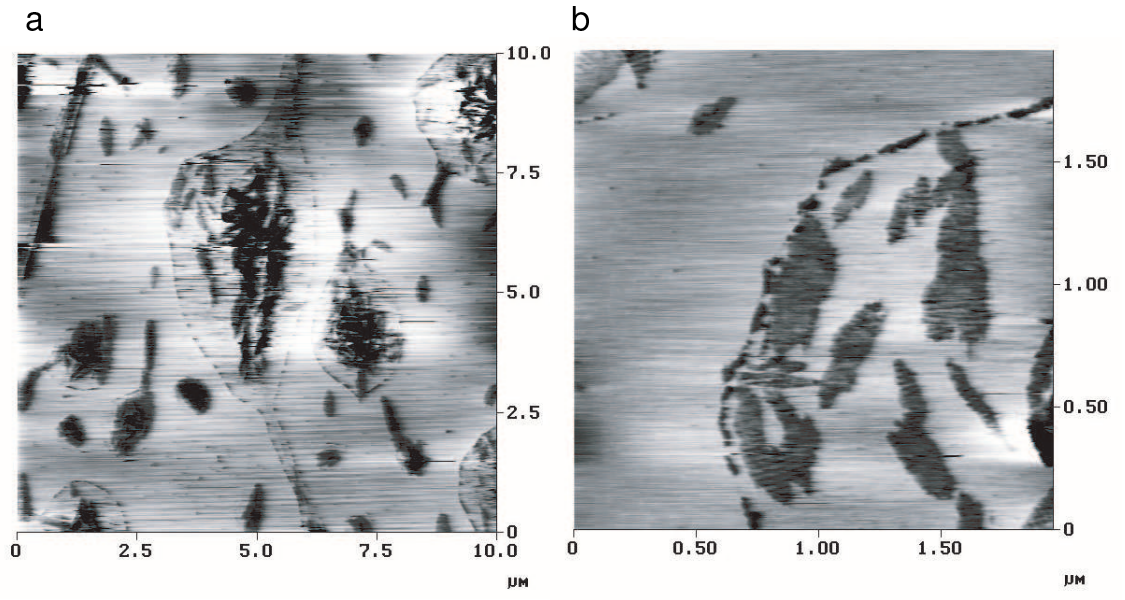


Figure 4.15: Debris apparently left by dewetting of the surface. Here, darker pixels represent greater height. (a) one area on a bar code lattice sample where capillary dewetting forces corralled DNA (and perhaps salt) into clumps. Ill-formed "rings" of debris surround these clumps (b) a close-up shows bar code lattice contained in the clump of debris.

loose weaves, sometimes leaving them intact. This point of view is supported by images like Figure 4.15 in which it appears that fragments of DNA lattice have been drawn up into fluid droplets that leave rings of debris upon drying. Within a single sample one often observes areas of good lattice preservation and, a few hundred microns away, an area of poor lattice preservation like that in Figure 4.15 (perhaps caused by incomplete drying of the sample). We also believe that rinsing (step 3) and flooding the sample with isopropanol (step 5) may cause some mechanical damage.

Mechanical damage to lattices is, theoretically, not a problem when one is interested in observing statistical features of the patterns in DNA lattices, as we are for the bar code tiles. The density of errors in the lattices should be unaffected by shredding of the

lattices into smaller lattices. However, the pattern of stripes near the edge of lattices can be difficult to interpret — the greater the fraction of edges in an image, the more data one may have to throw away. Also one might hypothesize that lattices shred preferentially at errors; then measuring the error rate in small lattices might yield an underestimate of the true error rate. Thus, in order to get good error statistics, we would like to be able to reproducibly observe relatively large bar code lattices (at least a micron per side). And, in order to create large scale patterns with our computational constructions, we would like to make very large DNA lattices, spanning many microns.

Nickolas Chelyapov suggested that the mechanical damage apparently due to pipetting might be avoided by annealing the DNA solution *in situ* over the mica surface. He suggested that we conduct the annealing in sealed chambers[50] designed for performing *in situ* PCR using a PCR machine designed for this purpose. We tried this technique, which replaces steps 1 and 2 of Winfree's procedure and often observed results like those shown in Figure 4.16 for striped tiles and Figure 4.6 and Figure 4.7 for bar code tiles. Strikingly, the entire surface of the mica was often covered with small seemingly interconnected domains of DNA lattice. For almost every sample observed, the first AFM field observed contained DNA lattice; serial searching over many 10 micron AFM fields was usually unnecessary. Lateral movement of the sample through an entire millimeter often resulted in similarly good results. We were disappointed that only small domains (a few hundred nanometers on a side) were observed since relatively large domains (a couple microns on a side) could often be observed by deposition of lattice from solution. Intriguingly, the small domains of lattice seem relatively well aligned (a small number of domains seemed to be present at

[50]Frame-seal chambers from MJ Research, Inc., Waltham, MA

orientations different from the primary alignment). We conjectured: (i) that after annealing, the entire surface of the mica had been covered by a single domain of DNA lattice (ii) that the fluid forces[51] involved in rinsing, drying, and applying isopropanol to the surfaced had ripped small sections out of the larger lattice and flipped some over to yield the images that we observed.

To test this hypothesis, we avoided steps 3–5 by imaging the DNA lattice under buffer using tapping mode AFM — tapping mode imaging applies much less lateral force on a sample and thus does not sweep DNA off a surface when it only loosely bound (as DNA is under buffer). As an added precaution, after removing the mica from its chamber, we added approximately 100 μL of annealing buffer with an extra 2 mM $NiCl_2$. Ni^{2+} ions enhance binding of DNA to mica ([HL96]) and facilitate imaging under buffer[52]. Figure 4.17 shows the result; unfortunately small domain size is still observed — it appears that the domain size observed under contact mode reflects the real distribution present under native conditions. A major difference is that the small domains extend until they touch each other — there are no holes in the surface coverage. A great benefit of this technique is its consistency — whenever we have obtained images in contact mode under buffer we have obtained essentially the same result.

[51] It is also possible the DNA changes from B-form to A-form when it is dehydrated by the isopropanol. This might introduce strain and cause lattice ripping.

[52] Initial attempts at imaging lattices under buffer without added $NiCl_2$ failed but, since these were also our first attempts at tapping mode imaging, we should again try imaging without $NiCl_2$. [HL96] states that Mg^{2+} alone, while sufficient to adhere DNA to mica for contact imaging under isopropanol, is insufficient for binding DNA to mica for tapping mode imaging under buffer. But perhaps DNA lattices will bind more tightly than the linear duplex DNA studied in [HL96].

Why are small domains observed when DNA lattices are formed over mica, but larger domains can be observed when DNA lattices are formed solution? We believe that the difference lies in the mechanism of nucleation. In solution lattices nucleate *homogeneously* — that is, DNA lattices nucleate exclusively by the binding of DNA tiles to other DNA tiles. Over a mica surface, we believe that DNA lattices nucleate *heterogeneously* — that is, DNA tiles stick to the mica surface, and the DNA lattices actually nucleate and grow on the surface. This may explain our observation that the lattices appear to be oriented — their orientation may be dictated by the first few tiles which may have an interaction with the underlying mica lattice. The small domain size may also be explained by lattice growth on the mica surface — perhaps independently nucleated DNA lattices grow until they run into each other and form domain boundaries. DNA lattices growing in solution are not similarly constrained. Thus to increase the size of DNA lattices grown on mica, we may have to decrease the rate of heterogeneous nucleation. Perhaps we can do this by reducing the stickiness of DNA for the mica; specifically we can reduce the concentration of divalent cation (Mg^{2+}) in the annealing buffer (Mg^{2+} is thought to bind both to the negatively charged mica surface and the negative backbone of DNA). Removing the Mg^{2+} entirely from the annealing buffer, however, abolishes the formation of DNA lattices — it decreases the melting temperature of the 5 base sticky ends below room temperature (20° C). Lattice formation is recovered by the addition of another 300 mM of monovalent ammonium ions (in the form of ammonium acetate, also determined by melting studies) so, in the future, we may use an ammonium acetate buffer in lieu of the magnesium-containing buffer normally used.

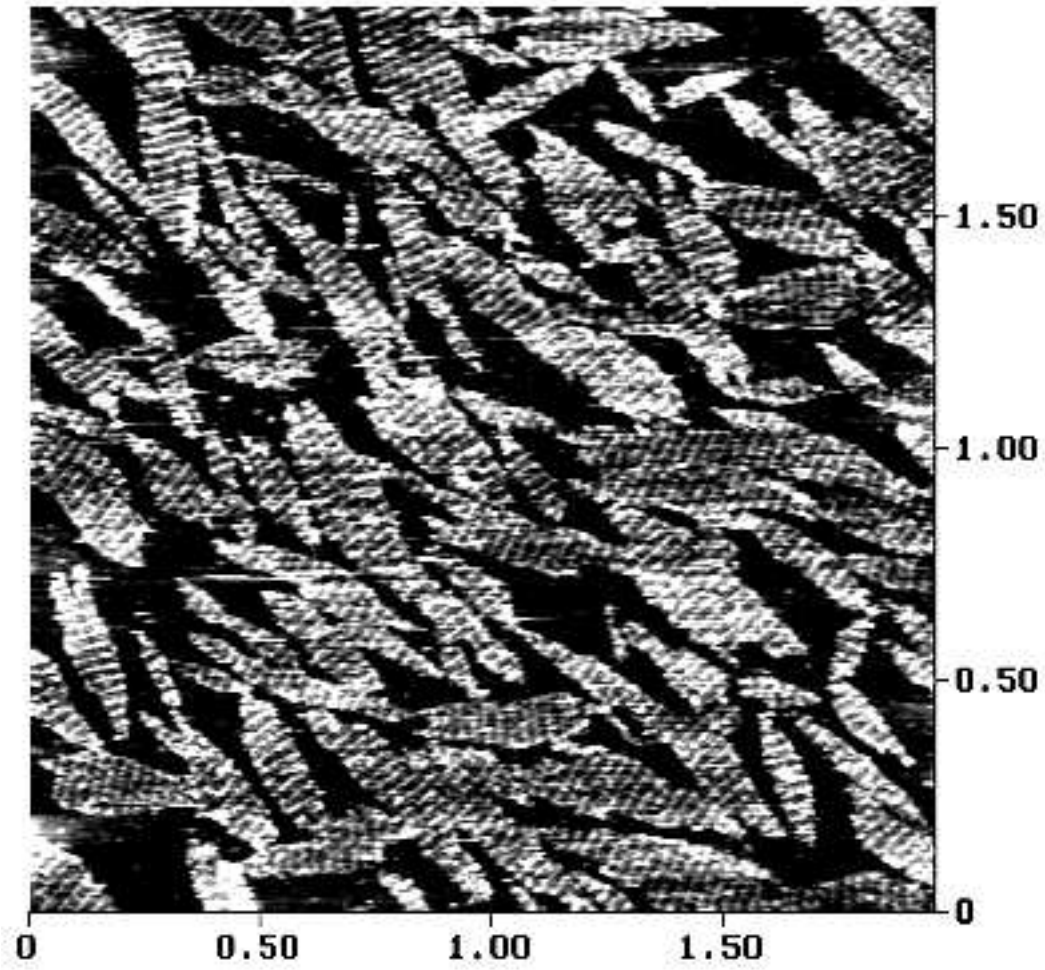


Figure 4.16: AFM image of striped DNA lattices under isopropanol. Here, lighter pixels represent greater height. Contact mode imaging of striped lattice (r and s tiles) annealed over mica. Imaging was performed under isopropanol after rinsing the mica with deionized water and drying using Dust-off brand "canned air". Similar results were obtained by drying under a stream of dry nitrogen.

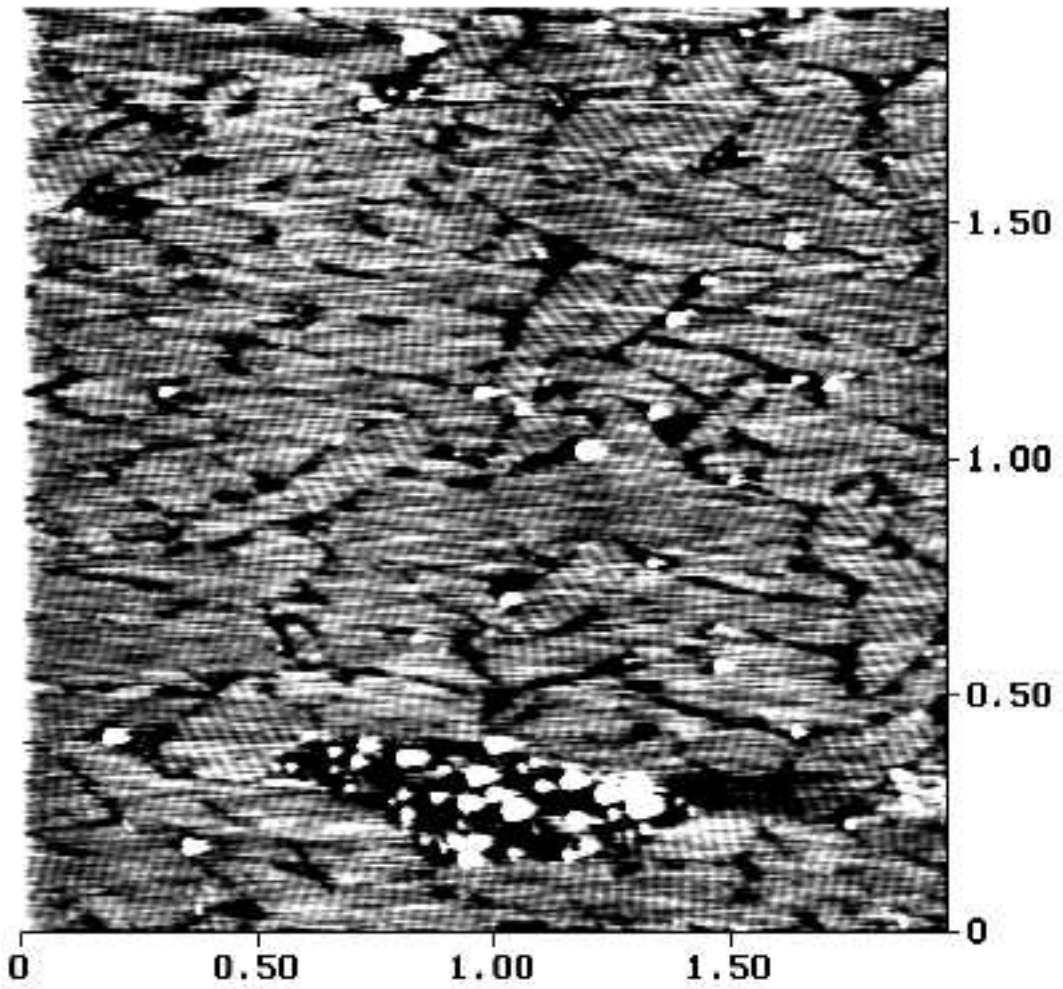


Figure 4.17: AFM image of striped DNA lattices under buffer. Here, lighter pixels represent greater height. Tapping mode images of the same lattices under standard buffer with 2 mM $NiCl_2$ added. While DNA appears to cover the surface completely, the same small domains (as in Figure 4.16 are observed and domain alignment is not entirely uniform.

Imaging in tapping mode is very promising and we have only just started to explore its uses. We have many experiments to perform, including imaging lattices prepared in PCR tubes under buffer — this eliminates steps 3–5 and, in contrast to *in situ* annealing over mica, may allow us to see the larger domains that we believe form in solution. Another thing we would like to do is make a "movie" of lattice formation; tapping mode imaging under buffer may allow us to mix pre-annealed DNA tiles at room temperature ($\sim 20^\circ$ C) and watch their association in real time. Alternatively, we may make use of a heated stage to perform the anneal *in situ* under the atomic force microscope.

Still, imaging under isopropanol in contact mode is, in many ways, preferable to imaging under buffer in tapping mode: contact mode imaging is easier to perform, faster, and yields, on average, higher resolution images. Hence, it is still of interest to image in contact mode under isopropanol — we just need to find a way to avoid the mechanical damage we believe is caused by dewetting. The question is, how best to get a sample of DNA sitting on mica under buffer to one under isopropanol without ever drying the sample out. One potential way of doing this is to use a dilution series; the sample is first dipped in de-ionized water[53] to eliminate salts and then in successively more concentrated isopropanol/water mixtures. Another way to execute the desired change is to first remove the salt[54] by dipping the sample in de-ionized water and then removing the water by freeze drying. Isopropanol has a low surface tension so adding the isopropanol for imaging after

[53]We suspect that gently adding even pure de-ionized water to the sample does not greatly disturb DNA on the surface since we have rinsed samples destined for tapping mode image with de-ionized water and observed results as good as those in Figure 4.17.

[54]This step is unnecessary if a volatile buffer that will be removed during freeze-drying can be used. Ammonium acetate buffer is such a buffer, but unfortunately must be used at concentrations so high that removing it completely is difficult.

freeze drying may cause little damage. So far, neither the dehydration series (of 5% steps) nor the freeze drying approach has yielded acceptable images.

Chapter 5

On Applying Molecular Computation to the Data Encryption Standard

This is sensitive stuff ... maybe we should encrypt it ... nah, too hard.

—*Rivest, Shamir, and Adleman*[1]

5.1 Abstract

Boneh, Dunworth, and Lipton have described the potential use of molecular computation in attacking the United States Data Encryption Standard (DES). Here, we[2] provide a description of such an attack using the *sticker model* of molecular computation. Our analysis suggests that such an attack might be mounted on a table-top machine, using approximately a gram of DNA and might succeed even in the presence of a large number of errors.

[1] A loose paraphrasing of a story by Len Adleman regarding the plans for an early hardware implementation of RSA.

[2] This chapter represents work published with Len Adleman, Sam Roweis, and Erik Winfree [ARRW99]. See Chapter 1 Section 1.6 for details.

5.2 Introduction

With their work on DES, Boneh, Dunworth, and Lipton [BDL96] provided the first example of a "practical" problem which might be susceptible to molecular computation. DES is one of the most widely used cryptographic systems. It produces a 64-bit ciphertext from a 64-bit plaintext under the control of a 56-bit key. While it has been argued that special purpose electronic hardware [Wie94] or massively parallel supercomputers might be used to break DES in a reasonable amount of time, it appears that today's most powerful sequential machines would be unable to accomplish the task. We continue the work of Boneh, *et al.* by considering the difficulty of breaking DES on the *sticker model* of molecular computation, recently proposed by Roweis *et al.* [RWB+98]. While our results are encouraging, it must be stressed that the feasibility of such an attack will ultimately be decided in the laboratory.

In this paper we consider the so called plaintext-ciphertext attack. Here the cryptanalyst obtains a plaintext and its corresponding ciphertext and wishes to determine the key used to perform the encryption. The most naive approach to this problem is to try all 2^{56} keys, encrypting the plaintext under each key until a key that produces the ciphertext is found. Remarkably, a significantly more efficient attack is not known and consequently this brute force approach will be the one considered here.

We begin by describing the algorithm to implement a plaintext-ciphertext attack for breaking DES at a logical level. This allows us to identify the fundamental operations we need to implement on a *stickers machine*, and serves as a roadmap for what follows.

5.3 The Molecular Algorithm

Start with approximately 2^{56} identical ssDNA *memory strands* [RWB+98] each 11580 nucleotides long. We think of each memory strand as containing 579 contiguous blocks (referred to as regions in [RWB+98]) $B_0, B_1, B_2, ..., B_{578}$ each 20 nucleotides long. As is appropriate in the sticker model there are 579 stickers $S_0, S_1, ..., S_{578}$ — one complementary to each block. (We refer to memory strands with annealed stickers as *memory complexes*.) Hence, we consider that each strand represents a 579-bit memory, and we sometimes use B_i to refer to the bit which B_i represents. Block B_0 is never set and is used later in the implementation of the algorithm (Subsection 5.4.1). Blocks B_1 though B_{56} of the memory strands are used to store a key, the next 64 blocks, $B_{57}, ..., B_{120}$, will eventually encode the corresponding ciphertext, and the remainder of blocks are used for intermediate results during the computation. The *stickers machine* which processes the memory strands to compute the ciphertexts does so under the control of a microprocessor. Because the plaintext is the same in all cases, the microprocessor may store it; we do not need to represent the plaintext on the memory strands. Now, given a plaintext-ciphertext pair, the algorithm is performed in three steps:

1. *Input step: Initialize* the memory strands to form memory complexes representing all 2^{56} keys.

2. *Encryption step:* On each memory complex *compute the ciphertext* corresponding to the encryption of the plaintext under that complex's key.

3. *Output step: Select* the memory complex whose ciphertext matches the given ciphertext, and *read* the corresponding key.

The bulk of the work is performed during the second step, where DES data encryption occurs, so we outline it below. Our interest is in demonstrating how DES can be implemented on a molecular computer, and for these purposes the exact details of DES are unnecessary. For details, see [oS77]. We will instead focus on the essential operations required in DES, and how these operations are combined to effect the full algorithm.

DES is a 16-round cipher. In each round a new 32 bit intermediate result is produced. These are designated $R_1, \ldots, R_{16}$. We store R_{15} and R_{16} in locations B_{57} through B_{120} (adjacent to the key), while $R_1, \ldots, R_{14}$ are stored in locations B_{121} through B_{568}. Essentially, R_{15} and R_{16}, taken together, form the desired ciphertext. (We encode the ciphertext adjacent to the key for implementation reasons explained in Subsection 5.4.4.) The left 32-bits and right 32-bits of the plaintext are referred to as R_{-1} and R_0, and are known to the controlling microprocessor.

Bits B_{569} through B_{578} are used as a *workspace* and are written and erased during the course of the computation. Hence, unlike the other bits which are used in a "write once" fashion, these bits may be cleared; for implementation reasons, we always clear the entire workspace at once.

Essentially, R_i is obtained from R_{i-1} and R_{i-2} by the following computation:

$$R_i = R_{i-2} \oplus S(E(R_{i-1}) \oplus K_i)$$

where $\oplus$ denotes exclusive or (x-or), K_i denotes a round dependent selection of 48 bits from the key, E denotes the *expand* function which takes the 32 bits of R_{i-1} and repeats or permutes them to yield 48 bits, and S denotes the *S-function* which takes a 48-bit input

and maps it to a 32-bit output. The function E, the function S and the selection K_i are *hard-coded*, like the plaintext, into the microprocessor.

In fact, the S-function can be separated into eight independent 6-bit to 4-bit functions known as *S-boxes*. Hence, each R_i may be computed in eight independent operations each of which produces a 4 bit *chunk* of the result. A given chunk is a function of 16 *input bits*: 6 bits of R_{i-1}, 6 bits of K_i and 4 bits of R_{i-2}. We describe the computation of a chunk below:

1. 6 bits of R_{i-1} and 6 bits of K_i are x-ored to produce a 6-bit result which is then stored in the workspace locations $B_{569}, \ldots, B_{574}$.

2. One of the S-box functions is applied to bits $B_{569}, \ldots, B_{574}$ and the 4-bit result is stored in the workspace locations $B_{575}, \ldots, B_{578}$.

3. Bits $B_{575}, \ldots, B_{578}$ are x-ored with 4 bits of R_{i-2} to produce the desired chunk of R_i which is then stored in the appropriate four blocks of the intermediate result bits $B_{57}, \ldots, B_{568}$.

4. If the chunk being computed is not the last chunk of R_{16}, the entire workspace, bits $B_{569}, \ldots, B_{578}$, is *cleared* in preparation for future use.

The positions on each memory complex of the 16 input bits required to compute a given chunk depend only on the chunk number (1,...,8) and the round number (1,...,16), though the 0/1 value of those bits will vary from memory complex to memory complex. The controlling microprocessor knows which positions contain these bits (they are hard-coded) and knows the x-or or S-box which it needs to apply.

We see, then, that encrypting a plaintext with DES comes down to a process of either (1) selecting 2 bits, producing their x-or, and writing the result in a new bit, or (2) selecting 6 bits, applying an S-box, and writing the resulting 4 bits.

5.4 Implementation

We now turn to implementing the algorithm on a stickers machine. Such a machine, as described in [RWB+98], may be thought of as a "parallel robotic workstation". It consists of a *rack* of tubes (*data tubes*, *sticker tubes*, and *operator tubes*), some *robotics* (arms, pumps, heater/coolers, connectors, etc.) and a microprocessor that controls the robotics. Roweis *et al.* assume that the components of the robotics and a set of three data or operator tubes may be arranged to perform any of the following four operations: *separate*, *combine*, *set* and *clear*.

We assume that the robotics are capable of an extended set of operations:

1. *Parallel separate.* The robotics can *separate* the DNA from each of 32 data tubes into two more data tubes by using 32 separation operator tubes at once.

2. *Parallel combine.* The robotics can *combine* the DNA from 64 different data tubes into one data tube at once. We assume that the *blank operator tube* used for a *combine* in [RWB+98] is really just a connector which is part of the robotics.

3. *Parallel set.* The robotics can, using one sticker tube with stickers S_k, *set* the bit B_k on the complexes in 64 different data tubes at once. We assume the *sticker operator tube* used for *set* in [RWB+98] is just a filter that can be built into the robotics.

4. *Clear.* The robotics can *clear* the workspace bits on all complexes in one data tube. We assume that the stickers on the workspace are removed simultaneously. Hence the workspace blocks may be implemented using so called *weak regions*, [RWB+98]. Again, we assume the *sticker operator tube* used for *clear* in [RWB+98] is just a filter that can be built into the robotics.

Hence, we perform the above four operations using just *data tubes* that may hold DNA memory complexes, *sticker tubes* that are (for the purpose of the computation) an inexhaustible source of a particular sticker S_k, and *separation operator tubes* that hold probes for a particular block B_k.

In the following subsections we describe, where applicable, the implementation of the molecular algorithm using these operations and, for the purposes of estimating the *time* and *space* required by a stickers machine we keep track of the following three *resource quantities*:

1. Total *steps.* We define the number of *steps* as the number of *parallel separations*, *parallel combines*, *parallel sets* or *clears* that any given complex experiences *after* it has been initialized. Hence, we count the operation of the robotics on a large number of tubes in parallel as a single step and ignore the (perhaps serial) process of moving data and operator tubes.

2. Total *rack tubes.* We define the number of *rack tubes* to be the total number of data tubes, sticker tubes, and separation operator tubes used during the computation. All of the tubes are reusable, so we only need copies of a tube if a particular kind of tube must be used more than once in a parallel operation. We note that we never

need duplicates of sticker tubes — our robotics are incapable of using more than one sticker tube at once. We will however need duplicate separation operator tubes and many data tubes since we often wish to separate complexes in several different data tubes on the same bit B_i at once.

3. Maximum number of *active tubes* per operation. We define the number of *active tubes*, for any time during the computation, as the number of tubes which the robotics have removed from the rack and are currently processing. Note that the maximum number of active tubes defines the width of the parallelism used by our algorithm and hence it must match the parallelism built into our robotics.

We note that these quantities are applicable for only for the *computation of the ciphertexts* (step 2 of the molecular algorithm) and the *selection* of the given ciphertext (the first part of step 3 of the molecular algorithm). These parts of the molecular algorithm require only the operations given above and hence are performed using the stickers machine. The *initialization* of the memory complexes and final *reading* of the key, however, use some operations that are not included above (*i.e.* dividing the contents of a tube into two tubes, PCR amplification, ligation, etc.) These special operations are used at most once during the implementation of the molecular algorithm and we assume that they could be performed by the human operator of the stickers machine in a reasonably short amount of time (a few hours) and in a small space (a few tubes). Hence they are ignored in the final discussion of the resources required by a stickers machine (Subsection 5.4.5).

5.4.1 Initialization of the memory strands

First we must create the initial tube encoding keys. Our desire is to have each of the 2^{56} memory strands store a different key. This might be accomplished, for example, in the following way:

1. Divide the memory strands into two tubes, A and B.

2. Add an excess of S_1 through S_{56} to tube A and allow them to saturate the first 56 blocks on each strand.

3. Use the complement of B_0 as a probe to *separate* the memory complexes in tube A from the excess stickers.

4. Add tube B to tube A.

5. Heat and cool tube A to randomly reanneal the stickers.

The memory complexes produced by this process appear to be reasonably modeled with a Poisson distribution; it is expected that approximately 63% of keys will be represented and that, on average, there is one of each key. Hence, if no errors are committed during the computation, we have a reasonable chance of recovering the key for a ciphertext of interest. Of course, the chance of succeeding can be increased by starting with more memory strands. To insure that approximately 95% of the keys are represented and that on average three copies of a key are present, we could simply use three times as much DNA. These issues are discussed in more detail in the Section 5.5.

5.4.2 Implementing the fundamental operations

As discussed in Section refmolecular-algorithm, the DES encryption algorithm is a composition of just two simple functions:

- x-ors which are 2-bit input to 1-bit output maps,
- S-boxes which are 6-bit input to 4-bit output maps.

The computation of the 2-bit to 1-bit x-or function is illustrated in Figure 5.1.

We now describe the computation of the x-or function $B_k = B_i \oplus B_j$ in greater detail, give the overhead required, and generalize this to an n-bit to m-bit function:

A. *Parallel separate* the sample two times to yield four data tubes, one for each possible value B_iB_j. This is accomplished by first using one separation operator tube specific for B_i and then, in parallel, using *two* separation operator tubes specific for B_j. (In Figure 5.1 the bits being considered are shaded gray.) Roweis *et al.* model a single separation as an operation involving three active tubes at once: a source data tube, a separation operator tube, and one of two destination data tubes picked up and filled by the robotics in sequence. Hence, during each single separation, three tubes are active at once and three data tubes are used. During the second *parallel separation* above, then, six data tubes are used and six tubes are active.

   For an n-bit to m-bit function this generalizes to:

   Parallel separate the sample n times to yield 2^n data tubes, one for each value of the n-bit input. This requires 2^{i-1} separation operator tubes for the ith *parallel*

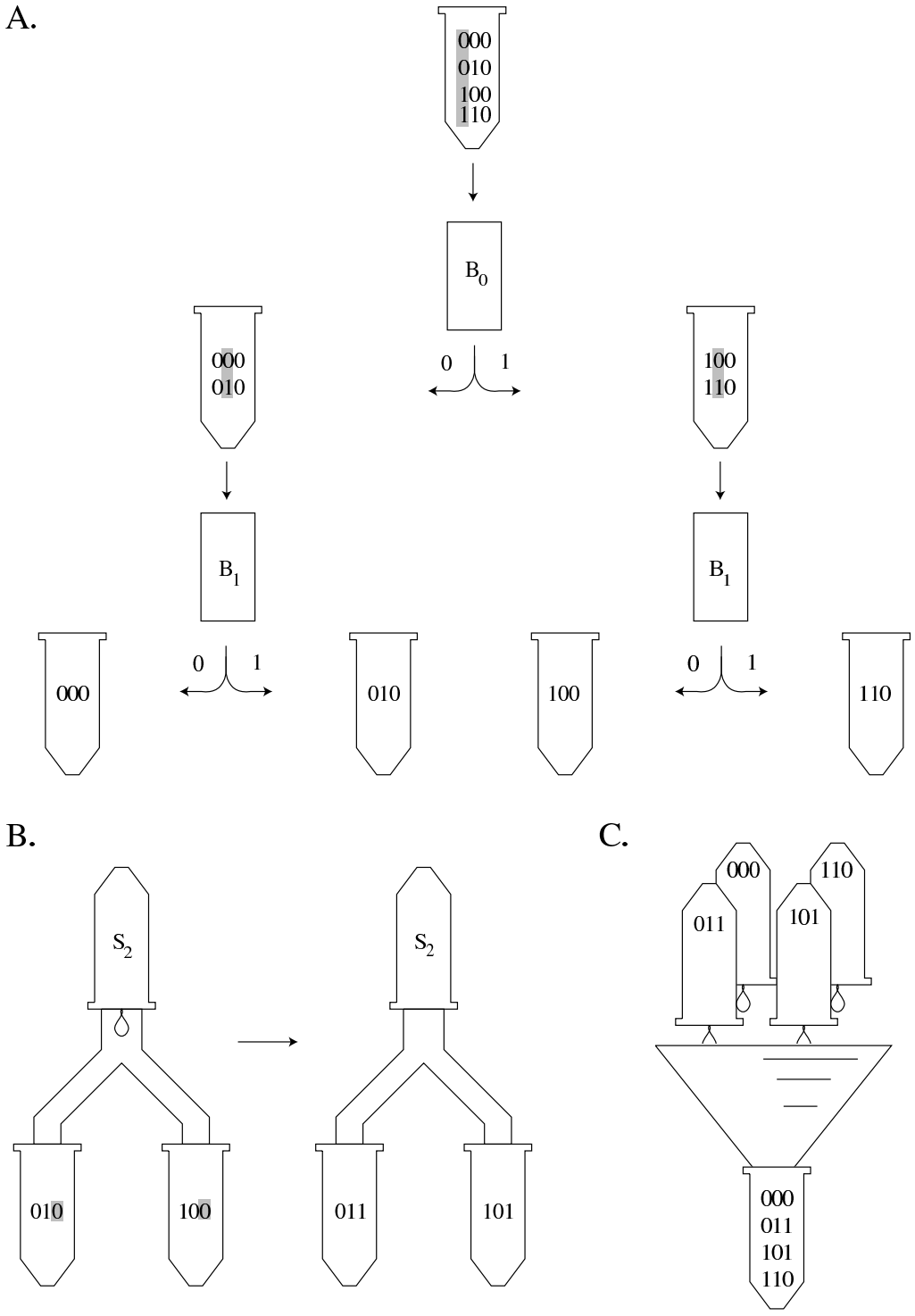


Figure 5.1: Computation of the 2-bit to 1-bit x-or function: $B_2 = B_0 \oplus B_1$.

separation (a total of $2^n - 1$). Hence for the nth parallel separation $3 \times 2^{n-1}$ data tubes are required and $3 \times 2^{n-1}$ tubes are active.

B. *Parallel set* B_k to 1 with an S_k sticker for all tubes for which this is applicable. For an x-or this is only applicable when $B_iB_j = 01$ or 10 but for a general 2-1 function this may require the addition of a sticker to any subset of the four data tubes in parallel. This requires one sticker tube and four data tubes, a total of five active tubes.

   For an n-bit to m-bit function this generalizes to:

   Parallel set a (possibly different) subset of the 2^n data tubes m times using a total of m sticker tubes. This requires $2^n + 1$ active tubes. Note that the subset of data tubes to which the *parallel set* is applied is determined solely by the algorithm stored in the microprocessor.

C. *Parallel combine* the contents of all four data tubes into one data tube. This requires five data tubes and hence five active tubes.

   For an n-bit to m-bit function this generalizes to:

   Parallel combine the contents of all 2^n data tubes into one data tube. This requires $2^n + 1$ data tubes and $2^n + 1$ active tubes.

At the end of our x-or operation, all of our DNA has been returned to a single tube.

In general, an n-bit to m-bit function requires $n + m + 1$ steps, $2^n - 1$ separation operator tubes (specific to various bits), m sticker tubes, a maximum of $3 \times 2^{n-1}$ data tubes, and a maximum of $3 \times 2^{n-1}$ active tubes.

At this point we can see that one of the resource quantities of interest, the maximum number of *active tubes* has already been specified. An 6-bit to 4-bit S-box is the biggest n-bit to m-bit function we implement in DES so we never use more than 96 active tubes. (The only other operation used in the algorithm, *clear*, uses but one active tube.)

The number of *rack tubes is* the sum of the data, sticker and separation tubes used but we may not simply sum up these numbers for an S-box and regard the result as its contribution to the computation. Data tubes are interchangeable so we know that the maximum number used in an S-box — 96 — will be the data tube contribution to the total *rack tube* count. However, sticker tubes and separator tubes have an identity, they are associated with a particular block of the memory complexes. For the calculation of *rack tubes* (and total steps as well) we need to consider more details of the molecular algorithm.

Thus, in the next subsection we consider how the x-ors and S-boxes used to compute DES are composed, count the total number of steps, and discuss ways to plan our use of separation operator tubes and sticker tubes to minimize the total number of rack tubes.

5.4.3 Computing the ciphertexts

We recall that the basic unit of computation in our molecular algorithm is a 4 bit *chunk* that is computed using a composition of x-ors and S-boxes which takes 16 bits of input. For any given chunk the positions of these 16 input bits are the same for *every* memory complex. Thus the positions of the input bits are *hard-coded* into the microprocessor and we ignore their exact values in what follows.

From our consideration of n-bit to m-bit functions, we know that 4 steps are required to compute an x-or and 11 steps are required to compute an S-box. From this it follows

that to compute a chunk requires $6 \times 4 + 11 + 4 \times 4 = 51$ steps. After a chunk is computed, if the workspace is to be used again, it must be *cleared*. To complete the computation requires computing $16 * 8 = 128$ chunks, and *clearing* the workspace 127 times; hence the total number of steps required is 6655.

During the computation of an x-or, one separation operator tube is required to *separate* on the first bit, and two separation operator tubes are required to *separate* (in parallel) on the second bit. Bits on the memory complex can be divided into equivalence classes based on their properties, for example whether they are part of the key or part of the work space. To economize on separation operator tubes when performing x-ors we choose to *separate* first on a bit belonging to larger equivalence class, and second on a bit belonging to a smaller equivalence class. Hence, for each x-or involving a bit of R_{i-2} (of which there are 448 possibilities from $R_1, \ldots, R_{14}$) and a bit from the workspace $B_{575}, \ldots, B_{578}$ (of which there are only 4), we *separate* on the R_{i-2} bit first and the workspace bit second. Likewise, for each x-or involving a bit of R_{i-1} (of which there are 480 possibilities from $R_1, \ldots, R_{15}$) and K_i (of which there are 56 possibilities), we *separate* on the R_{i-1} bit first and the key bit second. It follows that for each of the bits from $R_1, \ldots, R_{15}$ one separation operator tube specific to that bit is needed and, for each of the bits of K and each of the bits $B_{575}, \ldots, B_{578}$ two separation operator tubes specific to that bit are needed. Hence these bits require a total of $480 + 2 \times 56 + 2 \times 4 = 600$ separation operator tubes.

The implementation of the S-boxes demonstrates another way that we may economize on separation operator tubes: frequently used subcomputations do not require that we use up a new bit (and consequently another separation operator tube) every time they are run. Instead their input and output may be stored on a rewritable region of the memory

complex — hence our placement of the input to the S-boxes in the first six workspace positions and the output in the last four workspace positions. Because of this, even though there are eight different S-boxes (one for each chunk) they all may use the same separation operator tubes and sticker tubes (though, under control of the microprocessor, the stickers are applied differently for each S-box). The S-boxes, then, only require an additional 63 separation operator tubes to *separate* the DNA into all possible 6-bit strings $B_{569}, \ldots, B_{574}$.

Thus, to complete the ciphertext generation stage of the algorithm, we require 663 separation operator tubes. In all, $512 + 10 = 522$ sticker tubes are required to hold the stickers used to write the intermediate results and workspace bits. (Recall that our robotics only use one sticker tube at a time.) The greatest number of data tubes used during the computation is 96, during the final parallel *separation* step of an S-box computation. In total $663 + 512 + 96 = 1271$ rack tubes are required to compute the ciphertexts.

5.4.4 Selecting the given ciphertext and reading the correct key

Once the ciphertexts have been computed, the desired key can be *selected* by searching for the complex which has the given ciphertext encoded next to its key. This requires 64 *separation* steps. Upon isolating the desired complex, it is necessary to *read* its key. Reading could be attempted using single molecule detection and a binary tree decoding as described in [RWB+98]. However, it is not clear that such an approach can be satisfactorily carried out in lab. Below we describe two additional approaches, each of which entails some modification of the methods described in the previous sections:

- In the first approach, the memory strands are $5'$ biotinylated. Once the ciphertexts have been computed, as described in the previous sections, the memory complexes are transformed *en masse* into single-stranded form. One way to accomplish this is as follows:

  1. For each sticker $S_0, \ldots, S_{120}$ create a new *0-sticker*, S_i', which shares the $3'$ and $5'$ 8-mers with S_i but which differs from it in the middle 4-mer.
  2. Add an excess of the S_i' to the final solution under conditions which favor annealing despite the mismatches.
  3. Add ligase. Each memory complex now has a heteroduplex region in its key-ciphertext section where the non-memory strand consists of a sequence of (regular) stickers and 0-stickers. Note that our original decision to place the ciphertext next to the key on the memory strands is no longer a mystery. Our placement minimizes the number of ligations which must be successful for a strand which encodes both the ciphertext and key to form.
  4. *Separate* out the biotinylated memory strands using a streptavidin-coated solid support and retain the new strands as the *library* of solutions.

  This process essentially converts Sticker memory complexes into *Lipton style* memory strands [Lip95] for which each block has one of two unique sequences, one for 0 and another for 1. A ciphertext of interest may be *selected* (by applying the usual 64 *separation* steps), PCR amplified (using S_0' and the complement of the last ciphertext bit as primers), and *read* (with standard DNA sequencing).

Notice that, once created, the library of solutions can be replicated by PCR and hence multiple copies can be made. Each such copy is essentially a codebook consisting of (key, ciphertext) pairs. This codebook has approximately $2^{56} \times$ (56 key bits + 64 ciphertext bits) $= 2^{63}$ bits of information (the equivalent of approximately one billion 1 gigabyte CDs) but occupies a dry volume of approximately 1/7 of a teaspoon. As Boneh and Lipton have noted [BL95] such a codebook could be widely distributed and used to speed up subsequent attacks on DES.

- In the second approach to *reading*, instead of converting to Lipton style memory strands after the computation of ciphertexts, a Lipton-Sticker hybrid model for encoding the initial memory complexes is used.

    1. Make single-stranded DNAs representing all 2^{56} keys using the encoding of Lipton. Additionally, guarantee that each key strand begins and ends with the same short PCR primer.
    2. Ligate, to each of the Lipton key strands, identical 522 bit Sticker memory strands.
    3. Proceed to compute DES as usual.
    4. Perform 64 *separations* on the result to obtain the complex carrying the desired ciphertext.
    5. Perform PCR using the primers which bound the Lipton style key sequence. The key sequence will be amplified exponentially and may be sequenced.

We note that all of the techniques for *selecting* a desired ciphertext discussed here and in [RWB+98] would require 64 separation operations so we approximate that this

process requires an additional 64 steps. Further, we note that to effect all 64 separations an additional 32 separation operator tubes specific to the bits of R_{16} (in addition to the 32 separation operator tubes already counted for R_{15}) are required.

5.4.5 Discussion

In summary, to find the key for a DES-encoded plaintext-ciphertext pair, we first create memory complexes representing all keys, then we compute the ciphertext corresponding to each key (6655 steps), and finally we select and read the key of interest (64 steps). This requires a total of 6719 steps, each of which is one of the operations described above.

The actual running time for the algorithm depends on how fast the operations can be performed. If we assume, as we might if a graduate student had to perform each operation, that each operation requires 1 day, then the computation will require 18 years. If each operation requires 1 hour (Boneh *et al.* assume 2.4 hours) then the computation will require approximately 9 months. If each operation can be completed in 1 minute, perhaps using a robotic stickers machine, then the computation will take 5 days. Finally, if the effective duration of a step can be reduced to 1 second, perhaps by running the algorithm in a continuous flow *parallel refinery* [RW99], then the effort will require 2 hours.

The size of the rack is dictated by the amount of DNA used. When the 2^{56} memory complexes have half of their sticker positions occupied, as we expect will be the case at the end of the computation, they weigh approximately .7 g and, in solution at 5 g/liter, would occupy approximately 140 ml. Hence, the volume of the 1303 *rack tubes* (1271 for ciphertext computation, an additional 32 for ciphertext selection) need be no more than 140 ml each. It follows that the rack tubes occupy, at most, 182 L and can, for example, be

arrayed in a rack 1 m (approximately 39 inches) long and wide and 18 cm (approximately 7 inches) deep. Since the robotics must be able to operate on 96 *active tubes* in parallel we approximate the volume required by the robotics, give or take some arms and pumps, as 13 L — 1/14th the volume of the rack. The microprocessor is likely to be quite small. Thus, it is reasonable to assume the entire machine would fit on a desktop.

It is worth pointing out that, for much of the computation, the robotics are not processing at full capacity (96 tubes) and many of the tubes are sitting idle. Hence it may be possible to increase the parallelism significantly by pipelining the computation.

5.5 Analysis of Errors

We say that an error has occurred whenever a memory complex is transformed in an unintended way or ends up in an unintended place. Hence, there are many kinds of errors that can occur during the operation of our DNA computer: strands can break, stickers can fall off one memory complex and reanneal to another, complexes can be "lost" on the walls of a tube, complexes can end up in the wrong tube following a *separation*, etc.

For each operation, we define its error rate to be the fraction of molecules that commit an error during that operation. Some operations are more prone to errors than others. To simplify our analysis, we define E to be the error rate of the worst operation and assume that all of the operations have error rate E. Note that $1-E$ corresponds to what chemists call yield. Hence an error rate of 10^{-4} corresponds to a step yield of 99.99%.

Given an input ciphertext-plaintext pair, it is possible that several different keys map the ciphertext into the plaintext (though for DES, it seems unlikely that the number of such keys would be large). All such keys will be called *winning keys*. Under ideal conditions,

after the codebook is created and the separation on the input ciphertext performed, we are left with a *final tube* with the following properties:

1. For each winning key there is at least one complex encoding it.

2. All complexes that are there encode winning keys.

In reality this can fail for either of two reasons. First, complexes encoding winning keys may be missing, either because they were not created during initialization or because they encountered an error during the computation. Second, there may be *distractors*: complexes which do not encode winning keys, but due to errors end up in the final tube anyway. In subsection 4.1 we analyze the probability that a winning key has a complex encoding it in the final tube. In subsection 4.2 we calculate the expected number of distractors in the final tube.

We make the following assumptions:

1. After the initialization step, each complex encodes a 56-bit key chosen at random (*i.e.* chosen from the space of all 56-bit keys with equal probability).

2. DES with the input plaintext maps each of the keys to a random ciphertext.

3. A complex which encounters an error during the computation produces a random ciphertext (*i.e.* unrelated to the ciphertext normally associated with that complex's key).

5.5.1 Probability that a winning key has a complex encoding it in the final tube

In the computation above, we began with 2^{56} memory strands. It will be convenient to carry out the analysis in greater generality. We now assume that we begin with $2^{56}X$ memory strands, where X is a positive rational. Informally, X is the factor by which we multiply our original amount of DNA.

Let K_w be a winning key. Following initialization, the number of memory complexes which encode K_w is given by a binomially distributed random variable $(n, p) = (2^{56}X, \frac{1}{2^{56}})$.

The probability that a memory complex makes it correctly through all 6655 steps of the computation and 64 steps of the selection process (in total 6719 steps) is given by:

$$S = (1 - E)^{6719}$$

Hence, after the computation, the number of memory complexes in the final tube which encode K_w is given by a binomially distributed random variable $(n, p) = (2^{56}X, S\frac{1}{2^{56}})$. Because $2^{56}X$ is very large and $S\frac{1}{2^{56}}$ is very small, this distribution may be approximated by a Poisson distribution with Poisson parameter $\lambda = np = SX$. From this it follows that the expected number of complexes encoding K_w in the final tube is SX and that the probability that a complex encoding K_w is in the final tube is:

$$(1 - e^{-SX})$$

In particular when $X = 1/S$, the expected number of complexes encoding K_w in the final tube is one and the probability that a complex encoding K_w is in the final tube is 63%. We will refer to 63% as a *reasonable chance*.

5.5.2 Number of distractors in the final tube

For a memory complex M, let $H(M)$ denote the Hamming distance of the ciphertext encoded on M from the input ciphertext. For M to enter the final tube, $H(M)$ errors must occur during the final 64 separation steps. By our assumptions, after the computation of the codebook, each memory complex (whether it has encountered an error or not) encodes a ciphertext which is a random 64 bit string. It follows that the Hamming distances associated with memory complexes will be a binomially distributed random variable $(n, p) = (64, 0.5)$. Hence, the probability that $H(M) = L$ is:

$$\binom{64}{L}\left(\frac{1}{2}\right)^{64-L}\left(\frac{1}{2}\right)^{L} = \binom{64}{L}\frac{1}{2^{64}}$$

The probability of complex M with Hamming distance $H(M) = L$ making it through the 64 step selection process is given by the probability that it correctly negotiates $64 - L$ separations for which it matches the input ciphertext times the probability that it commits an error at the L separations for which it mismatches the correct ciphertext:

$$(1 - E)^{64-L}E^{L}$$

There are $2^{56}X$ complexes in the codebook so the expected number of distractors is given by:

achievable error rate E	0	10^{-4}	10^{-3}	10^{-2}
X	1	2	830	2.1×10^{29}
grams of DNA required	.7	1.4	580	1.5×10^{29}
distractors	.004	.008	3.2	8.3×10^{26}

Table 5.1: Quantity of DNA required and number of distractors as a function of error rate.

$$\sum_{L=0}^{64} \frac{2^{56} X}{2^{64}} \left(\begin{array}{c} 64 \\ \\ L \end{array} \right) (1-E)^{64-L} E^L = \frac{X}{256}$$

Perhaps surprisingly, the expected number of distractor molecules is independent of the error rate. Note, in particular, that for $X = 1$ the expected number of distractors is less than one.

5.5.3 Feasibility

Combining the results from Subsections 5.4.2 and 5.4.3 gives Table 5.1 which shows for various error rates, the amount of DNA that must be pushed through the DES computation to insure that there is a *reasonable chance* (63%) of getting a winning key in the final tube. Table 5.1 also records the expected number of distractors which will be present in the final tube.

This indicates that for an error rate of 10^{-4}, a little more that 1 gram of DNA is needed and the final tube will usually have no distractors. If an error rate of 10^{-3} is achievable, then less than a kilogram of DNA is needed and only a small number of distractors need be dealt with before finding the correct answer. However, if an error rate of 10^{-2} is the best that is attainable, then huge amounts of DNA are needed (approximately 23 Earth masses) to have a *reasonable chance* that a winning key ends in the final tube, but even then it will

have to be distinguished from a colossal number of distractors - clearly an unacceptable situation. In such cases, where only very high error rates are possible, techniques like those described in [RW99] (for example, a *refinery algorithm*) may be used to reduce the amount of DNA required. [RW99] gives a brief analysis of the application of a refinery algorithm to DES.

5.6 Conclusions

We wish to emphasize that our description of an attack on DES is, at this point, entirely theoretical and whether it can be carried out in the lab remains to be seen. Huge challenges remain. For example, as yet, we have been unable to perform *separations* (or any of the sticker operations) in the lab with error rates approaching 10^{-4}. Nonetheless, the analysis presented in this paper demonstrates (at least in principle) two things:

- "Real problems" can be solved with small machines which do not require huge amounts of DNA and use little or no enzymes.
- Error rates similar to those normally demanded of electronic computers are not required.

If the attack on DES described here can be carried out in the lab, then some other cryptosystems might also be vulnerable to this approach. Indeed, the small size of the machine we describe suggests that systems like the 64-bit key FEAL cryptographic system of Shimizu-Miyaguchi [SM88] might be susceptible to such an attack.

Finally, there are several messages for cryptography in these findings. First, it seems appropriate to reconsider one of the "axioms of cryptography": Improvements in computational power always favor the cryptographer over the cryptanalyst. This is almost certainly untrue. The analysis presented here suggests the possibility of computers with super-parallelism that can help the cryptanalyst immensely, yet provide no help for the cryptographer. Even if DNA computers prove infeasible, it is possible that new machines machines capable of super-parallelism may make cryptosystems like DES insecure.

The DNA computer is an example of a super-parallel machine with very slow processors (complexes of DNA). The potential vulnerability of DES arises for two reasons. First, the key space is insufficiently large. Second, the DES algorithm is "too short." The fact that only 6655 steps are needed to do an encryption allows the slow DNA processors to finish their encryptions in (at least in theory) a reasonable amount of time. Hence the much valued throughput speed of DES and similar systems, may carry with it a potential vulnerability to super-parallel machines with slow processors.

5.7 Coda

DNA computing has often seemed like an answer in search of a problem — for researchers in DNA computation this phenomenon has become known as the "search for the killer app". Of all the strictly computational applications suggested as a killer app for DNA computation, the cryptanalysis of DES seemed one of the most plausible and promising. One of the few problems of general interest that was "wide" in space and "shallow" in depth it seemed ideally suited for taking advantage of the parallelism of DNA computing.

Breaking DES with DNA computation would not have changed the world[3] but it would have shown DNA computation to be similar in power to electronic computers, an amazing feat considering the maturity of our computer technology. Back when this chapter was originally written, in 1996, an expert at Sandia National Laboratories estimated that the world's fastest general-purpose massively-parallel supercomputer (Intel's "Tflops", a 9200 Pentium Pro machine at Sandia) could break DES with an expected time of about a month. Our best guess of how long it might take a DNA computer to break DES was comparable so, if we had had such an DNA computer in hand, DNA and silicon computers might have competed.

Computers are now approximately 10 times faster and the expected time to crack DES using a similarly-sized parallel supercomputer is down to about 3 days. Further, a number of highly publicized "DES challenges" by RSA Security have been successfully met by distributed computing on the internet or special purpose hardware. First, an organization of computers on the internet, distributed.net, cracked DES using background cycles in 39 days[4]. Next, the Electronic Frontier Foundation build a special purpose DES hardware cracker "Deep Crack" for under $250,000 [TG98] that, in July of 1998 broke DES in 56 hours. Finally, in January of 1999, the combined efforts of distributed.net and Deep Crack broke DES in 23 hours. Those few who still considered DES secure back in 1996 certainly no longer consider it so now.

In the meantime, progress on DNA computing has been slow; we have reported the solution a six-variable satisfiability problem [BJR+01] and have recently completed the

[3]Strong encryption, presumed unbreakable by a conventional computer DNA or silicon, has been available for 20 years in the form of RSA.

[4]http://www.distributed.net/des/

solution of a 20-variable problem. This computation, on a combinatorial library of 20-bit strings, requires fewer separations than the computation of a single S-box.[5] And the computation requires no stickers — we have not yet begun to explore the sticker part of the Sticker model. To implement the cryptanalysis of DES with DNA as outlined in this paper, while perhaps possible, would require a major effort costing millions of dollars and many man years; I hesitate to guess an upper limit but I cannot imagine it taking less than $10 million and 50 man-years (We have spent approximately 5 man-years and $1 million to get where we are, and I would judge that we are less than a tenth of the way to solving DES). To summarize, breaking DES has become irrelevant and we are nowhere close to breaking DES with DNA computation.

What then is the significance of this chapter? I think this work still stands as a good treatment of the application of DNA computation to a relatively difficult computational problem, of real world interest, using a very concretely specified model of DNA computation. It gives actual numbers for the length of the DNA molecules required, the number of steps required and number of grams of DNA required (given reasonable assumptions) to break DES. In computer science we often eschew such specific values for complexity measures, favoring instead asymptotic expressions. But in the real world constants matter; specifically, the decision not to invest significant resources in DNA computation have rested on these constants. From our analysis of DES and the state of silicon and DNA technologies one may (and I do) conclude two things: (i) DNA and DNA computation are amazing — with much less DNA than is in the human body it is plausible that we

[5]The 20-variable problem required 24 separations. An S-box requires separation on 6 bits but 32 distinct separation operations must be successful (even though many may be carried out in parallel).

could solve a computational problem as difficult as DES. (ii) It is not worth pursuing the solution of DES in the lab.

I think more may be concluded, however. The solution for DES we present using the Sticker model, and the Sticker model in general, seem to use DNA fairly efficiently (*e.g.* storing bits at a density within a factor of 40 from the maximal density possible using DNA). It does not seem likely that other models of computation will allow us to do *more difficult* computations than DES with the same or fewer DNA resources. Thus, barring an unexpected breakthrough, it is not worth pursuing the Sticker model or other related models of DNA computation.

Where does this leave DNA computation? While surely not in competition with electronic computers, DNA computation may find use in applications for which electronic computers may be used only with great difficulty. For example, DNA computations embedded in chemical systems may be able to directly control the chemical system in complicated ways; an electronic controller would require some interface to the chemical world. Or DNA computation may be used in the self-assembly of complicated patterns; again an electronic computer would need some interface to the physical world. Happily, the last possibility is what the rest of this thesis is about!

Reference List

[ACGH01] Leonard Adleman, Qi Cheng, Ashish Goel, and Ming-Deh Huang. Running time and program size for self-assembled squares. *Symposium on the Theory of Computing (STOC 2001)*, 2001.

[Adl79] Leonard M. Adleman. Time, space, and randomness. *MIT Report LCS/TM-131*, April 1979.

[Adl94] Leonard M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, November 11, 1994.

[Adl00] Leonard M. Adleman. Toward a mathematical theory of self-assembly. *USC Technical Report*, 2000.

[ARRW99] Leonard M. Adleman, Paul W. K. Rothemund, Sam Roweis, and Erik Winfree. On applying molecular computation to the data encryption standard. *Journal of Computational Biology*, 6(1):53–63, 1999.

[BCGW99] N. Bowden, I.S. Choi, B.A. Gryzbowski, and G.M. Whitesides. Mesoscale self-assembly of hexagonal plates using lateral capillary forces: Synthesis using the "capillary bond". *Journal of the American Chemical Society*, 121(23):5373–5391, 1999.

[BDL96] Dan Boneh, Christopher Dunworth, and Richard J. Lipton. Breaking DES using a molecular computer. In Lipton and Baum [LB96], pages 37–65.

[Ben85] Charles H. Bennett. Information, dissipation, and the definition of organization. In D. Pines, editor, *Emerging Syntheses in Science*, pages 297–313, Sante Fe, NM, 1985. Sante Fe Institute.

[Ben95] Charles H. Bennett. Logical depth and physical complexity. In Rolf Herken, editor, *The Universal Turing Machine A Half-Century Survey*, pages 207–235, Wien, 1995. Springer-Verlag.

[Ber66] Robert Berger. The undecidability of the domino problem. *Memoirs of the AMS*, 66:1–72, 1966.

[BJR+01] Ravinderjit S. Braich, Cliff Johnson, Paul W. K. Rothemund, Darryl Hwang, Nickolas Chelyapov, and Leonard M. Adleman. Solution of a satisfiability problem on a gel-based DNA computer. In Anne Condon and Grzegorz Rozenberg, editors, *DNA Computing 6th International Workshop on DNA-based computers, DNA 2000, Leiden, the Netherlands, June 2000*, volume 2054 of *Lecture Notes in Computer Science*, pages 27–42, Berlin, 2001. Springer.

[BL95] Dan Boneh and Richard J. Lipton. Batching DNA computations. *Princeton University Computer Science Tech-Report, CS-TR-489-95*, 1995.

[BM92] M. Bousquet-Mélou. Convex polyominoes and heaps of segments. *J. Phys. A: Math. Gen.*, 25:1925–1934, 1992.

[Bon88] John Tyler Bonner. *The Evolution of Complexity by Means of Natural Selection.* Princeton University Press, Princeton, New Jersey, 1988.

[BTCW97] N. Bowden, A. Terfort, J. Carbeck, and G.M. Whitesides. Self-assembly of mesoscale objects into ordered two-dimensional arrays. *Science*, 276:233–235, 1997.

[CdE01] Qi Cheng and Pablo Moisset de Espanes. Resolving two open problems in the self-assembly of squares. *unpublished manuscript*, 2001.

[CG95] A. R. Conway and A. J. Guttman. On two-dimensional percolation. *Journal of Physics A*, 28:891–904, 1995.

[Cha77] G. Chaitin. Algorithmic information theory. *IBM Journal of Research*, 21:350–359, 1977.

[DDYN94] A.S. Dimitrov, C.D. Dushkin, H. Yoshimura, and K. Nagayama. Observations of Latex Particle Two-Dimensional-Crystal Nucleation in Wetting Films on Mercury, Glass, and Mica. *Langmuir*, 10:432–440, 1994.

[DiV90] D.P. DiVincenzo. Physical models of perfect quasicrystal growth. In Tolédano [Tol90], pages 133–139.

[DS95] S. Dworkin and J.I. Shieh. Deceptions in quasicrystal growth. *Communications in Mathematical Physics*, 168(2):337–352, 1995.

[DYP95] A. D. Dinsmore, A. G. Yodh, and D. J. Pine. Phase-diagrams of nearly hard-sphere binary colloids. *Physical Review E, Part B*, 52(4):4045–4057, October 1995.

[Els85] V. Elser. Comment on: "Quasicrystals: A new class of ordered structure". *Physical Review Letters*, 54:1730, 1985.

[Fin86] Robert Finn. *Equilibrium Capillary Surfaces.* Springer-Verlag, New York, 1986.

[FLS65] Richard P. Feynman, Robert B. Leighton, and Matthew Sands. *The Feynman Lectures on Physics.* Addison-Wesley, Reading, Massachusetts, 1965.

[Gar77] Martin Gardner. Mathematical games: Extraordinary nonperiodic tiling that enriches the theory of tiles. *Scientific American*, pages 110–121, Jan 1977.

[GL87] A. Garg and D. Levine. Faceting and roughening in quasicrystals. *Physical Review Letters*, 59:1683–1686, 1987.

[GN96] J. Gaydos and A.W. Neuman. Line tension in multiphase equilibrium systems. In A.W. Neuman and J.K. Spelt, editors, *Applied Surface Thermodynamics*, pages 169–238, New York, 1996. Marcel Dekker.

[GS87] Branko Grünbaum and G. C. Shephard. *Tilings and Patterns.* W. H. Freeman and Company, New York, 1987.

[GTB+00] David H. Gracias, Joe Tien, Tricia L. Breen, Carey Hsu, and George M. Whitesides. Forming electrical networks in three dimensions by self-assembly. *Science*, 289:1170–1172, August 18, 2000.

[Han74] William Hanf. Nonrecursive tilings of the plane I. *The Journal of Symbolic Logic*, 39:283–285, 1974.

[HL96] Helen G. Hansma and D. E. Laney. DNA binding to mica correlates with cationic radius: assay by atomic force microscopy. *Biophysical Journal*, 70(4):1933–1939, 1996.

[HSM96] K. Hosokawa, I. Shimoyama, and H. Miura. Two-Dimensional Micro-Self-Assembly using the Surface Tension of Water. *Sensors and Actuators A*, 57:117–125, 1996.

[IK80] Shôkichi Iyanaga and Yukiyosi Kawada, editors. *Encyclopedic Dictionary of Mathematics.* MIT Press, Cambridge, 1980.

[KB91] Arthur Kornberg and Tania A. Baker. *DNA Replication (2^{nd} ed.).* W. H. Freeman and Company, 1991.

[KBJ83] M. Kolb, R. Botet, and R. Jullien. Scaling of kinetically growing clusters. *Physical Review Letters*, 51:1123–1126, 1983.

[KN94] P. A. Kralchevsky and K. Nagayama. Capillary Forces between Colloidal Particles. *Langmuir*, 10:23–26, 1994.

[KPDN95] P. A. Kralchevsky, V. N. Paunov, N. D. Denkov, and K. Nagayama. Stresses in Lipid Membranes and Interactions between Inclusions. *J. Chem. Soc. Faraday Trans.*, 91:3415–3432, 1995.

[KR73] D. A. Klarner and R. L. Rivest. A procedure for improving the upper bound for the number of n-ominoes. *Canadian Journal of Math*, 25:891–904, 1973.

[KRYP94] P. D. Kaplan, J. L. Rouke, A. G. Yodh, and D. J. Pine. Entropically driven surface phase-separation in bindary colloidal mixtures. *Physical Review Letters*, 72(4):582–585, 1994.

[Lau70] R. A. Laudise. *The Growth of Single Crystals.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1970.

[LB96] Richard J. Lipton and Eric B. Baum, editors. *DNA Based Computers: Proceedings of a DIMACS Workshop, April 4, 1995, Princeton University*, volume 27 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, Providence, Rhode Island, 1996. American Mathematical Society.

[Lev84] L.A. Levin. Randomness conservation inequalities: Information and independence in mathematical theories. *Information and Control*, 61:15–37, 1984.

[Lip95] Richard J. Lipton. DNA solutions of hard computational problems. *Science*, 268(5210):542–545, 1995.

[LL00] Michail G. Lagoudakis and Thomas H. LaBean. 2-D DNA self-assembly for satisfiability. In Winfree and Gifford [WG00], pages 141–154.

[LNP92] Igor Lyuksyutov, A. G. Naumovets, and V. Pokrovsky. *Two-dimensional crystals.* Academic Press, Inc., San Diego, 1992.

[LS84] D. Levine and P.J. Steinhardt. Quasicrystals: A new class of ordered structures. *Physical Review Letters*, 53:2477–2480, 1984.

[LV77] L. A. Levin and V. V. V'jugin. Invariant properties of informational bulks. In *Lecture Notes in Computer Science 53*, pages 359–364, New York, 1977. Springer-Verlag.

[LV97] Ming Li and Paul Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications (Second Edition).* Springer Verlag, New York, 1997.

[LWR00] Thomas H. LaBean, Erik Winfree, and John H. Reif. Experimental progress in computational by self-assembly of DNA tilings. In Winfree and Gifford [WG00], pages 123–140.

[Mac95] A.L. Mackay. Generalised crystallography. *Journal of Molecular Structure (Theochem)*, 336:293–303, 1995.

[Mar95] Ivan V. Markov. *Crystal Growth for Beginners: fundamentals of nucleation, crystal growth, and epitaxy.* World Scientific, Singapore, 1995.

[Mea83] P. Meakin. Formation of fractal clusters and networks by irreversible diffusion-limited aggregation. *Physical Review Letters*, 51:1119–1122, 1983.

[MLRS00] 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(6803):493–496, 2000.

[Mor92] K. Morita. Computation-universality of one-dimensional one-way reversible cellular automata. *Information Processing Letters*, 42:325–329, 1992.

[Mye74] Dale Myers. Nonrecursive tilings of the plane II. *The Journal of Symbolic Logic*, 39:286–294, 1974.

[oS77] National Bureau of Standards. Data encryption standard. In *FIPS, pub. 46.* U.S. Department of Commerce, 1977.

[OSDS88] G.Y. Onoda, P.J. Steinhardt, D.P. DiVincenzo, and J.E.S. Socolar. Growing perfect quasicrystals. *Physical Review Letters*, 60:2653–2656, 1988.

[PB99] Steven Pigeon and Yoshua Bengio. Binary pseudowavelets and applications to bilevel image processing. *Data Compression Conference (DCC '99)*, pages 364–373, 1999.

[PBRPC98] C. Puente-Baliarda, J. Romeu, R. Pous, and A. Cardama. On the behavior of the sierpinski multiband fractal antenna. *IEEE Transactions on Antennas and Propagation*, 46(4):517–524, 1998.

[Pen78] Roger Penrose. Pentaplexity. *Eureka*, 39:16–22, 1978.

[Pen89] Roger Penrose. Tilings and quasicrystals: a non-local growth problem. In Marko Jarić, editor, *Introduction to the Mathematics of Quasicrystals*, pages 53–80, San Diego, CA, 1989. Academic Press.

[Pen91] Roger Penrose. *The Emperor's New Mind.* Penguin Books, New York, 1991.

[PKA69] W.K. Pratt, J. Kane, and H.C. Andrews. Hadamard transform image coding. *Proceedings of the IEEE*, 57(1):58–68, 1969.

[Pta92] Mark Ptashne. *A Genetic Switch, 2nd ed.* Cell Press & Blackwell, 1992.

[Rad62] Tibor Rado. On non-computable functions. *Bell System Technical Journal*, 41(3):877–884, May 1962.

[RBJ90] M. Ronchetti, M. Bertagnolli, and M.V. Jarić. Generation and dynamics of defects in two-dimensional quasicrystals. In Tolédano [Tol90], pages 141–158.

[Rei99] John Reif. Local parallel biomolecular computing. In Harvey Rubin and David Harlan Wood, editors, *DNA Based Computers III: DIMACS Workshop, June 23-25, 1997*, volume 48 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science.*, pages 217–254, Providence, RI, 1999. American Mathematical Society.

[Rob71] Raphael M. Robinson. Undecidability and nonperiodicity of tilings of the plane. *Inventiones Math.*, 12:177–209, 1971.

[Rot00] Paul Wilhelm Karl Rothemund. Using lateral capillary forces to compute by self-assembly. *Proceedings of the National Academy of Sciences*, 97:984–989, 2000.

[RW89] J.S. Rowlinson and B. Widom. *Molecular Theory of Capillarity.* Clarendon Press, Oxford, 1989.

[RW99] Sam Roweis and Erik Winfree. On the reduction of errors in DNA computation. *Journal of Computational Biology*, 6(1):65–75, 1999.

[RW00] Paul Wilhelm Karl Rothemund and Erik Winfree. The program size complexity of self-assembled squares. *Symposium on the Theory of Computing (STOC 2000)*, 2000.

[RWB+98] Sam Roweis, Erik Winfree, Richard Burgoyne, Nickolas V. Chelyapov, Myron F. Goodman, Leonard M. Adleman, and Paul W. K. Rothemund. A sticker-based model for DNA computation. *Journal of Computational Biology*, 5(4):615–629, 1998.

[Saf94] Samuel A. Safran. *Statistical Thermodynamics of Surfaces, Interfaces, and Membranes.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.

[SBGC84] D. Schectman, I. Blech, D. Gratias, and J.W. Cahn. Metallic phase with long-range orientational order and no translational symmetry. *Phys. Rev. Lett.*, 53:1951–1953, 1984.

[SCH+95] Willem P. C. Stemmer, Andreas Crameri, Kim D. Ha, Thomas M. Brennan, and Herbert L. Heyneker. Single-step assembly of a gene and entire plasmid from large numbers of oligodeoxyribonucleotides. *Gene*, 164(1):49–53, 1995.

[See98] Nadrian C. Seeman. DNA nanotechnology: novel DNA constructions. *Annual Review of Biophysics and Biomolecular Structure*, 27:225–248, 1998.

[Sen90] Marjorie Senechal. *Crystalline symmetries: an informal mathematical introduction.* Adam Hilger, New York, 1990.

[Sen95] Marjorie Senechal. *Quasicrystals and geometry.* Cambridge University Press, Cambridge, 1995.

[SM88] A. Shimuzu and S. Miyaguchi. Fast data encipherment algorithm FEAL. In *Lecture Notes in Computer Science 304*, pages 267–278, New York, 1988. Springer-Verlag.

[Smi97] Warren D. Smith. Computational complexity of synthetic chemistry — Basic facts. *NECI Technical Report*, April 18, 1997.

[SWY+98] N. C. Seeman, H. Wang, X. P. Yang, F. R. Liu, C. D. Mao, W. Q. Sun, L. Wenzler, Z. Y. Shen, R. J. Sha, H. Yan, M. H. Wong, P. Sa-Ardyen, B. Liu, H. X. Qiu, X. J. Li, J. Qi, S. M. Du, Y. W. Zhang, J. E. Mueller, T. J. Fu, Y. L. Wang, and J. H. Chen. New motifs in DNA nanotechnology. *Nanotechnology*, 9(3):257–273, 1998.

[Tay25] Wilson Taylor. *A new view of surface forces.* University of Toronto Press, Toronto, 1925.

[TG98] The Electronic Frontier Foundation and John Gilmore. *Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design.* O'Reilly and Associates, Cambridge, 1998.

[Tol90] J.-C. Tolédano, editor. *Geometry and Thermodynamics*, New York, 1990. Plenum Press.

[Wan61] Hao Wang. Proving theorems by pattern recognition. II. *Bell System Technical Journal*, 40:1–42, 1961.

[Wan63] Hao Wang. Dominoes and the AEA case of the decision problem. In Jerome Fox, editor, *Proceedings of the Symposium on the Mathematical Theory of Automata*, pages 23–55, Brooklyn, New York, 1963. Polytechnic Press.

[WG00] Erik Winfree and David K. Gifford, editors. *DNA Based Computers V: Proceedings of the* 5^{th} *DIMACS Meeting on DNA Based Computers, held at the*

Massachusetts Institute of Technology, June 14-15, 1999, volume 54 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, Providence, Rhode Island, 2000. American Mathematical Society.

[WH19] R. S. Willows and E. Hatschek. *Surface tension and surface energy and their influence on chemical phenomena.* J. & A. Churchill, London, 1919.

[Wie94] M. Wiener. Efficient DES key search. *Carleton University School of Computer Science Tech Report, TR-244*, 1994.

[Win96] Erik Winfree. On the computational power of DNA annealing and ligation. In Lipton and Baum [LB96], pages 199–221.

[Win98a] Erik Winfree. *Algorithmic Self-Assembly of DNA.* PhD thesis, California Institute of Technology, Computation and Neural Systems Option, 1998.

[Win98b] Erik Winfree. Simulations of computing by self-assembly. In Lila Kari, Harvey Rubin, and David H. Wood, editors, *Proceedings of the* 4^{th} *DIMACS Meeting on DNA Based Computers, held at the University of Pennsylvania, June 16-19, 1998*, preliminary, 1998.

[Win00] Erik Winfree. Algorithmic self-assembly of DNA: Theoretical motivations and 2D assembly experiments. *Journal of Biomolecular Structure & Dynamics*, pages 263–270, 2000. Special issue S2.

[WLWS98a] Erik Winfree, F. Liu, L. A. Wenzler, and Nadrian C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394:539–544, 1998.

[WLWS98b] 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.

[WMS91] George M. Whitesides, John P. Mathias, and Christopher T. Seto. Molecular self-assembly and nanochemistry: a chemical strategy for the synthesis of nanostructures. *Science*, 254:1312–1319, November 29, 1991.

[Wol94] Steven Wolfram. *Cellular Automata and Complexity.* Addison-Wesley, New York, 1994.

[WYS98] Erik Winfree, Xiaoping Yang, and Nadrian C. Seeman. Universal computation via self-assembly of DNA: Some theory and experiments. In Eric B. Baum and Laura F. Landweber, editors, *DNA Based Computers II: DIMACS Workshop, June 10-12, 1996*, volume 44 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science.*, pages 191–213, Providence, RI, 1998. American Mathematical Society.

[YH97] R.K.R. Yarlagadda and J.E. Hershey. *Hadamard Matrix Analysis and Synthesis With Applications to communications and Signal/Image Processing.* Kluwer Academic Publishers, Boston, 1997.

Appendix A

Self-similar transforms

In the introduction we gave two examples of interesting patterns that, given $\mathcal{T} = 2$ conditions, could be self-assembled with a small number of tiles: the binary counter (Figure 1.1) and the Sierpinski triangle (Figure 1.5). Both of these patterns could be interpreted as matrices and, given suitable decoration with logic elements, could be used to create circuits implementing their respective matrix transforms. In particular we outlined how the binary counter could be used to implement a demultiplexer (Figure 1.3), a useful circuit; whether the Sierpinski triangle has a useful circuit interpretation is not known.

Interestingly, both the binary counter and Sierpinski triangle are self-similar. It is often the case that simple 1D cellular automata generate self-similar patterns, and the translation of these cellular automata to small sets of self-assembling tiles is straightforward; many self-similar patterns seem easy to self-assemble. Thus an interesting question is whether there exist easily self-assembled self-similar patterns, other than the binary counter, whose interpretation as transforms yield some useful computation. We suggest

that the self-similar patterns underlying binary pseudowavelet matrices or Hadamard matrices, both mathematically useful transforms, might be self-assembled[1]. In Figure 1.6 we give a visual comparison of these matrices to the Sierpinski matrix. To help understanding of their relationship at a deeper level, below are their matrix definitions.

The binary pseudowavelet matrix is given by:

$$W_2 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad W_{2n} = \begin{bmatrix} T_n & W_n \\ W_n & 0 \end{bmatrix} \quad n = 2^k, k = 1, 2, \ldots$$

where T_n is the $n \times n$ matrix with its first row filled with ones and the rest of the matrix is filled with zeros, i.e.:

$$T_2 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, \quad T_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The Sierpinski matrix is given by:

$$S_2 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad S_{2n} = \begin{bmatrix} S_n & S_n \\ S_n & 0 \end{bmatrix} \quad n = 2^k, k = 1, 2, \ldots$$

The Hadamard matrix is given by:

[1]We have been collaborating on this problem with Matt Cook and have constructed tile sets (although we have not proven their correctness) for both patterns.

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad H_{2n} = \begin{bmatrix} H_n & H_n \\ H_n & -H_n \end{bmatrix} \quad n = 2^k, k = 1, 2, \ldots$$

For two-valued matrices A and B where one of the values is equal to 1, we say that $A \subset B$ iff $A_{ij} = 1 \implies B_{ij} = 1$ and $\exists (i,j)$ such that $B_{ij} = 1$ and $A_{ij} \neq 1$. For all $k \geq 2$ it is easy to show that $W_n \subset S_n \subset H_n$.

How can one use such transforms? We give one simple example and defer the reader to [PB99, PKA69, YH97] for others. Both the binary pseudowavelet matrix and the Hadamard matrix are useful for processing a signal, for example an image, so that it is suitable for "multi-resolution reconstruction". We are all familiar with multi-resolution reconstruction — when an image is downloaded over a slow connection, we see it first at low resolution, then at higher resolution, and then finally at its highest resolution. How might this be done? Consider a 1D black and white image that can be represented as a binary vector $\vec{v}$. Then we can represent $\vec{v}$ using the rows of the pseudowavelet matrix W_n as a basis, we perform the matrix multiplication $W_n^{-1}\vec{v}$ with multiplication replaced by AND and addition replaced by XOR and call the result $\vec{v}'$. If we inspect the rows of W_n we find that each row is composed of a single sequence of 1's of length 2^k for some integral k — a row of W_n can be thought of as representing a localized feature of a particular size, a larger feature if it has a longer row of 1's, a small feature otherwise. The bits of $\vec{v}'$ represent $\vec{v}$ decomposed over localized features of different sizes.

Thus we can then send $\vec{v}'$, one bit at a time, sending the bits that represent large features first, and bits representing small features last. On the receiving end, as bits come in, we reconstruct an approximation of $\vec{v}$ — $\vec{v}_k$ — by multiplying W_n by a vector that has

the k bits of $\vec{v}$ received so far, plus 0's where bits have yet to be sent. Our reconstructions $\vec{v}_k$ are successively higher resolution approximations of the vector $\vec{v}$ until finally, $k = n$ and $\vec{v}_k = \vec{v}$.

What difficulties are there in using self-assembly create circuits implementing such transforms? The reconstruction pseudowavelet transform W_n has a simple self-similar structure for which it seems likely we can find a simple self-assembly program. Unfortunately, the forward transform W_n^{-1} does not have a self-similar structure and thus seems more difficult to self-assemble (although reordering of the transforms might yield a pair $\dot{W}_n$ and $\dot{W}_n^{-1}$ that are both easy to self-assemble). The Hadamard matrix H_n can be used in similar ways and fortunately, up to a constant, the Hadamard matrix is its own inverse.

More difficult is the problem that the row orderings for the pseudowavelet transform and the Hadamard transform, as we present them, are not very useful. For example, in the self-similar versions of the pseudowavelet transform that we present, the transform has large-feature rows interleaved with small-feature rows in a complicated pattern. More useful is a transform that has the rows ordered based on the size of their features (in the case of the pseudowavelet transform) or their *sequency*, (in the case of Hadamard transforms; sequency, analogous to frequency, is the number of alternations between 0 and 1 in a row.) We may, or may not be able to come up with self-assembly programs for such ordered transforms. If we cannot, we may have to resort to additional self-assembled transforms that reorder the inputs or outputs for the Hadamard or pseudowavelet transforms.

Appendix B

On capillary forces, surface tension, and interfacial energy

Before performing the experiments described in Chapter 3, I never really understood surface tension or capillary action. I could never understand why a water strider can zip across the water without falling in nor why the paper towel Bounty is the "quicker-picker-upper", nor, most importantly, that both were facets of exactly the same phenomenon. I had read about the concepts of surface tension and capillary action in basic science textbooks as a kid no fewer than a dozen times. The oft-repeated concept that the surface of the water could be viewed as a "skin" or rubber sheet was, I believe, one source of my confusion. Why should the molecules at the surface of the water stick together any more tightly than molecules below them in the bulk? If water molecules were diffusing within the surface, or worse diffusing to and from the surface, what was providing the lateral restoring force between them, the "surface tension" provided by elastic polymer molecules in a rubber sheet?

In fact while both the surface of water and a rubber sheet have surface tension, the microscopic mechanism creating the surface tension is completely different. We can treat the molecules of a rubber sheet as a network of springs. Thus, for rubber sheets, we

naturally measure surface tension in units of force/unit length (say dynes/cm) — we can calculate surface tension by planning a cut in the sheet and adding up the force contributed by each spring that would be severed by the cut. For a fluid, we naturally measure surface tension in different units, energy/unit area (say ergs/cm^2) and we actually call this measurement a surface energy, or if there are two fluids like water and air (or one fluid and a solid) an interfacial energy.

The interfacial energy is a measure of potential energy — in particular, the potential energy of molecules at the interface. Consider a water molecule away from the interface in the *bulk*; it is hydrogen bonded to other water molecules in an ever changing pattern but we can picture it as, on average, bonded on all sides — let us say that it makes six bonds. When the molecule of water is taken from the bulk and added to the interface, it can no longer bond on one side reducing the effective number of bonds to five — we can say that a bond bond has been broken — the enthalpy of that bond is added to energy of the interface. Also, the entropy of a molecule changes when it moves from the bulk to an interface — in the 3D bulk it has more degrees of freedom than at the 2D interface. Thus it takes a certain free energy to create a patch of interface between water and air. Formally, the two units of surface tension, force/length and energy/area are completely interconvertible[1] and one can proceed merrily along treating a water/air interface as if

[1]See [RW89], pp. 5–6, for proof of equivalence for these two measures. Good mathematical treatments of capillarity are to be found in [Saf94] and [Fin86]. As is often the case, the most interesting prose descriptions of capillary phenomena are to be found in older books, for example [WH19] and [Tay25]. I find it useful to read books written before all the kinks in a theory have been worked out — the authors are often confused in the same ways I am confused. Also, the Feynman Lectures, volume II, p. 12-5 has an interesting treatment of rubber sheets as an analogy for electrostatics [FLS65]; this means that capillary forces are governed by equations analogous to those of electrostatics.

it were a rubber sheet. Conceptually, however, in light of its microscopic description, describing the surface of water as a skin or rubber sheet seem very inappropriate.

Armed with an understanding of surface tension as an interfacial energy we are now able to explain surface tension based phenomena. The water strider does not stay afloat because he is standing on a rubber trampoline; instead, he stays afloat because for him to be wet by water would require an increase in the water's surface area, and hence the total interfacial potential energy. Many water-going insects and spiders avoid being wet by water by sporting tiny hairs that increase their surface area. For the same reason water balls up and slips off the leaf of a Lamb's ear plant.

The sign of an interfacial energy need not be positive — sometimes increasing the area of an interface is a energetically favorable change. The classic textbook example of this is water's meniscus in a glass graduated cylinder; water creeps up the walls *against gravity* because the water molecules have such a favorable interaction with a glass surface. Likewise, Bounty is "the quicker-picker-upper" because water molecules have more favorable interactions with the paper towel molecules than they do with themselves. Thus when water molecules *increase* their area of contact with the paper towel by spreading throughout the paper towel, energy is released. The greater the surface area of the paper towel, the more water it can accommodate this way; thus we might explain Bounty's reputation as "the quicker-picker-upper" by assuming that the surface of Bounty paper towels is more fractal in nature than that of other paper towels. In effect, Bounty paper towels would have more surface area per "macroscopic measurable area" of paper towel.

Interfacial energy is thus the unifying concept for all these phenomena. One can't describe the interface between a fluid and a rigid solid as a rubber sheet but one can

measure its area. The absorbency of a paper towel cannot be understood by any kind of rubber sheet analogy but it can be explained by a negative interfacial energy.

Appendix C

Vertex stars in Penrose tilings

One way to define a Penrose tiling is by giving its tiles (a fat and a thin rhomb) and its matching rules (here the light gray and dark gray markings on the tiles, Figure C.1).

A Penrose tiling is defined as any tiling of the plane[1], by fat and thin rhombs, for which the matching rules are satisfied. Another way to define a Penrose tiling is by its vertex stars. The eight vertex stars for a Penrose tiling are given in Figure C.2.

A Penrose tiling is then defined as any tiling of the plane, by fat and thin rhombs, that contains these eight vertex stars and *only* these eight vertex stars.

Just as the Penrose tiles in a mathematical Penrose tiling occur in a definite ratio, so too do the vertex stars occur in characteristic proportions. Upon observing the self-assembly of the physical Penrose tiles I constructed, I noticed that one vertex star, the moon,

[1] A tiling is a complete covering of the plane, without gaps or overlaps.

Fat Rhomb (FR) Thin Rhomb (TR)

Figure C.1: The Penrose rhombs. Dark and light gray markings indicate matching rules.

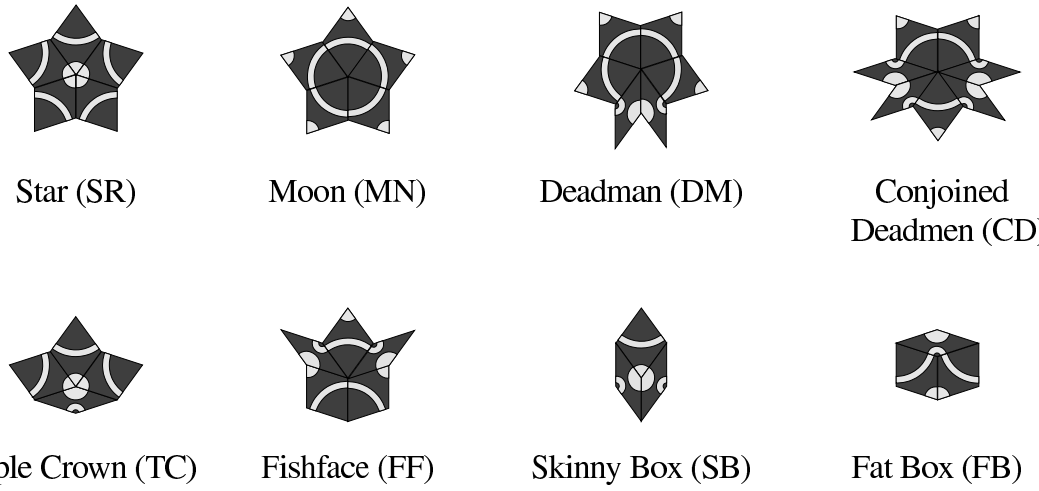


Figure C.2: The eight vertex stars that define a mathematical Penrose tiling. A Penrose tiling contains these eight vertex stars and *only* these eight vertex stars. While standard names exist for vertex stars of Penrose kites and darts, no standard names seem to exist for the vertex stars of Penrose rhombs; thus the names given here are peculiar to this thesis. "Deadman" is due to Len: "See, he has been shot in the neck..."

seemed very underrepresented in the aggregates formed — especially when compared to its dual vertex star, constructed from the same five tiles, the star. Before attempting any explanation, in order to determine whether this observation was significant, I had to know the theoretical ratio of stars to moons in a mathematical Penrose tiling. I asked Marjorie Senechal, an expert on Penrose tilings, where I might look up the answer. She didn't know and to my knowledge no one else has bothered to calculate it. For what its worth, I outline how to do it below.

We start with the question of how one can compute the ratio of fat rhombs to thin rhombs in a Penrose tiling following the method outlined in [Sen95], and generalize it to finding the ratio of the the Penrose vertex stars. To compute the ratio of fat rhombs to thin rhombs, we examine a recursive method for generating Penrose tilings, the inflation

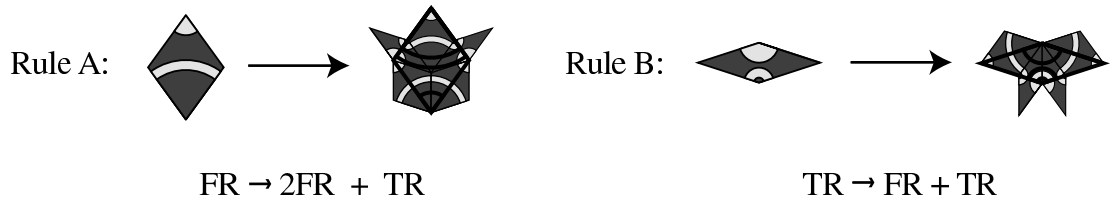


Figure C.3: Geometrical substitution rules for the inflation method of generating Penrose tilings. Note that the substitution rules, as normally drawn, depict only the interior of the substituted tile within the bold black outline. Here, for clarity of presentation, partial tiles have been extended beyond the outline of the original substituted tile. Below each rule we give a relation expressing the number of tiles of each type generated by the rule. When counting the number of tiles (or later vertex stars) generated by such a substitution we count only tiles or parts of tiles that fall within the outline of the original tile.

method. Given any patch of matched Penrose tiles P (a single tile is sufficient) the inflation method works as follows:

1. Subdivide each fat rhomb according to the the geometrical rule A in Figure C.3.

2. Subdivide each thin rhomb according to the geometrical rule B in Figure C.3.

3. Scale the patch by the golden ratio, $\tau = 1.618\ldots$

To tile the plane, these three steps must be repeated an infinite number of times; call each repetition of these steps a *stage* of the algorithm. Call the number of fat rhombs in a patch of Penrose tiles at the ith stage of this algorithm, FR_i. Call the number of thin rhombs in a patch of Penrose tiles at the ith stage of this algorithm TR_i. Then, by inspecting the geometrical substitution rule in Figure C.3 we can find out how a tile at stage i contributes to the populations of tiles at state $i+1$. For example, for Rule A we write $\mathrm{FR} \to 2\mathrm{FR} + \mathrm{TR}$ to indicate that "a fat rhomb at stage i yields 2 fat rhombs and 1 thin rhomb at stage $i+1$". Similarly, for Rule B we write $\mathrm{TR} \to \mathrm{FR} + \mathrm{TR}$. Putting these

two relations together we can see that the population vector of thin and fat rhombs, x_i obeys the recursion:

$$x_{i+1} = \begin{bmatrix} \mathrm{FR}_{i+1} \\ \\ \mathrm{TR}_{i+1} \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{FR}_i \\ \\ \mathrm{TR}_i \end{bmatrix} = Ax_i$$

Because the matrix A is a primitive matrix (it has all nonnegative integer values and there exists a k such that A^k has all positive values) the Pierre-Frobenius theorem applies. This theorem states that for all matrices B with all positive values, there exists a real eigenvalue w_1 such that $|w_1| > |w_j|$ for $j = 2 \ldots k$ where $w_2 \ldots w_k$ are the other eigenvalues of B. As the geometrical substitution rule above is iterated, the population of rhombs changes according to the recurrences written in the matrix equation above. Because A (which shares its eigenvectors with A^k) has an eigenvalue of greatest magnitude w_1, the population vector x_i eventually points in the direction of W_1 (*i.e.* there is a single dominant eigenvector, the so-called "lead eigenvector"). We thus define a vector x_∞ to be the limit of x_i normalized so that the proportion of thin rhombs is 1; x_∞ gives us the asymptotic proportion of fat and thin rhombs. As argued above, x_∞ is equal to the lead eigenvector W_1 of A normalized so that the proportion of thin rhombs is 1. We find that the eigenvalue $w_1 = \tau^2$ and:

$$x_\infty = \lim_{i \to \infty} \frac{1}{\mathrm{TR}_i} \begin{bmatrix} \mathrm{FR}_i \\ \\ \mathrm{TR}_i \end{bmatrix} = \begin{bmatrix} \tau \\ \\ 1 \end{bmatrix}$$

Thus the ratio of fat rhombs to thin rhombs in a Penrose tiling[2] is $\tau : 1$ — this is the stoichiometry with which I combined the physical Penrose tiles described in Chapter 3.

To extend the above analysis to Penrose vertex stars, the transformations of all eight vertex stars under the substitution rules A and B are diagrammed in Figure C.4. The transformations of Penrose tiles are included because without them counting the evolution of vertex stars under inflation becomes less straightforward. Given a Penrose tile or vertex star X at stage i of the inflation algorithm, call those tiles or vertex stars that fall within the boundary of X at stage $i+1$ the children of X.[3] Because vertex stars at stage i are composed of tiles at stage i, and each tile participates in four vertex stars, a particular structure at stage $i+1$ could be the child of four vertex stars and a tile from stage i. To avoid over-counting structures at stage $i+1$ due to "multiple parentage" we organize the accounting of structures as follows: (i) examine structures diagrammed in Figure C.4 from top to bottom and left to right. (ii) for each structure at stage i count, for inclusion on the right hand side of that structure's $\rightarrow$ ("yields") relation, all children except those already recorded on the right hand side of $\rightarrow$ relations for structures previously evaluated. For

[2]While the inflation method allows one to build arbitrarily large patches of matched Penrose tiles, it does not converge to a single Penrose tiling. Patches at odd stages are consistent with each other, patches at even stages are consistent with each other, but patches from even and odd stages are not consistent. In a sense, the inflation method generates two different Penrose tilings. More surprising is the fact that the Penrose tiles actually admit an *uncountable* number of distinct tilings modulo congruence [Sen95]. Despite this, however, it can be shown that the ratio of fat and thin rhombs is the same for *all* Penrose tilings [GS87] — exactly the ratio found for a Penrose tiling generated by inflation. We do not recount the proof in detail, but rather give the essential ideas. There exists a unique inverse map (call the process of iterating it *deflation*) for the geometric substitution given in Figure C.3. An arbitrary Penrose tiling P can be deflated N times to yield another Penrose tiling P_N. And P_N can be inflated N times to yield P. But we know that each tile of P_N inflates to a patch of tiles in P with a ratio of approaching that given by x_∞ as N gets large. Thus the ratio of fat and thin rhombs in P approaches the same limit. An almost identical proof shows that the ratios of vertex stars found for Penrose tilings generated by inflation, generalizes to *all* Penrose tilings.

[3]We count partial tiles inside the boundary as children of X; when we count them, however, we count them as a fraction of a tile. We consider vertex stars as "inside the boundary" if the vertex of the vertex star is inside the boundary; if parts of the tiles that make up a vertex star lie outside the boundary, we still count the vertex star as single vertex star.

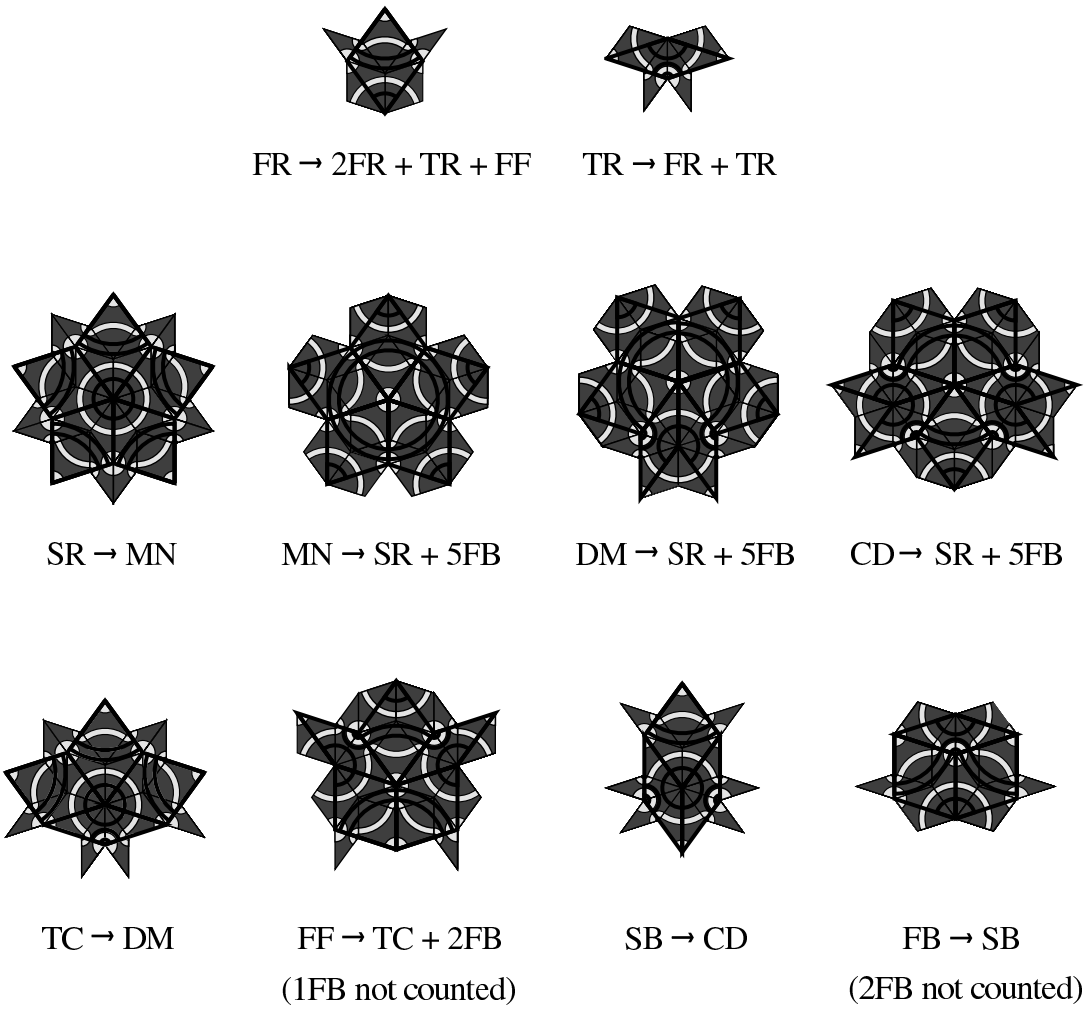


Figure C.4: Application of the inflation rules to the vertex stars. $\rightarrow$ relations are given for each of the Penrose tiles and vertex stars at stage i of the inflation method are given. Where it is not obvious, notes indicate that certain vertex stars should not be counted. $\rightarrow$ relations for FR and TR are included for mathematical convenience.

example, the vertex star FF is counted towards the $\rightarrow$ relation for FR but not where it occurs as a child in all subsequently evaluated structures. More subtly, 5 FB are credited for each of the vertex stars MN, DM, and CD but not where they occur later as children of the vertex stars FF or FB. Thus, evaluating the structures in a different order would give different $\rightarrow$ relations.

Each $\rightarrow$ relation under a structure in Figure C.4 becomes a column in a 10×10 matrix describing the evolution of tile and vertex star frequencies:

$$A = \begin{array}{c} \begin{array}{cccccccccc} \text{FR} & \text{TR} & \text{SR} & \text{MN} & \text{DM} & \text{CD} & \text{TC} & \text{FF} & \text{SB} & \text{FB} \end{array} \\ \begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 5 & 5 & 5 & 0 & 2 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{c} \\ \text{FR} \\ \text{TR} \\ \text{SR} \\ \text{MN} \\ \text{DM} \\ \text{CD} \\ \text{TC} \\ \text{FF} \\ \text{SB} \\ \text{FB} \end{array}$$

To find the population vector for this matrix we find the lead eigenvector[4] and normalize it so the number of thin rhombs is 1. This yields the population vector:

[4]To do so I used Matlab to compute the eigenvectors numerically, guessed symbolic values for the elements of the lead eigenvector, and checked them by computing the eigenvalue.

$$x_\infty = \lim_{i\to\infty} \frac{1}{\mathrm{TR}_i} \begin{bmatrix} \mathrm{FR}_i \\ \mathrm{TR}_i \\ \mathrm{SR}_i \\ \mathrm{MN}_i \\ \mathrm{DM}_i \\ \mathrm{CD}_i \\ \mathrm{TC}_i \\ \mathrm{FF}_i \\ \mathrm{SB}_i \\ \mathrm{FB}_i \end{bmatrix} = \begin{bmatrix} \tau \\ 1 \\ 1/\tau^2(1+\tau^2) \\ 1/\tau^4(1+\tau^2) \\ 1/\tau^5 \\ 1/\tau^4 \\ 1/\tau^3 \\ 1/\tau \\ 1/\tau^2 \\ 1 \end{bmatrix}$$

As before the eigenvalue for this eigenvector is τ^2. That the matrix derived for all the vertex stars gives the ratio of fat boxes to thin rhombs as one serves as a check of its validity — it is easy to inspect the Penrose tiles and convince oneself that there is a one to one correspondence between fat boxes and thin rhombs. The ratio of stars to moons, for which we went through this trouble, is found to be $\mathrm{SR} : \mathrm{MN} = \tau^2 : 1 \approx 2.6 : 1$. The ratio observed experimentally was quite different — 19 : 1; see Chapter 3, p. 160 for discussion of this discrepancy.

Finally, I note that there exist vertex stars that one can construct from the Penrose rhombs that satisfy the matching rules but are not part of any Penrose tiling, Figure C.5; they are known as *forbidden* vertex stars.

I never observed any of these vertex stars at any time during the self-assembly (they were not present in final structures but because I did not take video of the assembly and

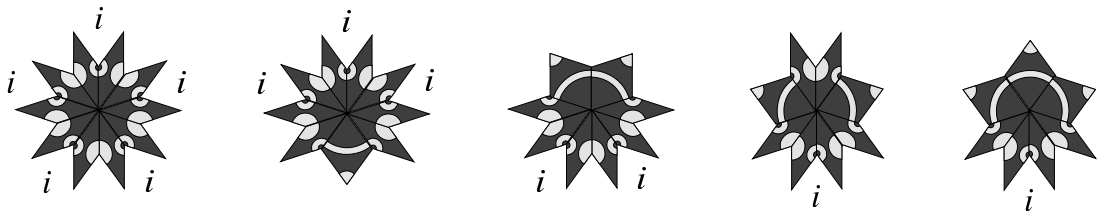


Figure C.5: Five forbidden vertex stars. Vertex stars exist that satisfy the Penrose matching rules but do not occur in a mathematical Penrose tiling. Surprisingly, none of these vertex stars were observed during self-assembly experiments.

inspect it as carefully as I did for XOR tiles, some of these vertex stars may have occurred at a low rate during assembly). This seems, at first, a bit strange since they are not disallowed by the local matching rules and the physical system knows nothing about the global pattern which it is supposed to construct.

I offer two possible explanations: (i) The forbidden vertex stars are "high energy structures" — that is, the ratio of unsatisfied bonds (the length of their perimeter) to the number of tiles (their area) is large — 2 : 1. Our experimental system evolves to minimize the interfacial potential energy and decrease the number of unsatisfied bonds. The allowed vertex stars have the same ratio of unsatisfied bonds to number of tiles but they can be combined into larger structures that satisfy more bonds and decrease the total interfacial energy. The forbidden vertex stars, because of their incompletable sites, cannot be incorporated into larger structures as low in energy as those formed by the allowed vertex stars. This does not explain why the forbidden vertex stars were never observed during assembly but explains why they should be rare among the final structures. (ii) The forbidden vertex are *a priori* unlikely because each one is composed of fat and thin rhombs in a ratio of less than 1:1. The tiles were mixed in a ratio of $\tau : 1$ so the collision of rarer

thin rhombs to form a structure enriched for thin rhombs should be unlikely. Two of the observed vertex stars do have ratios less than 1:1, SB (ratio 1:2) and CD (ratio 3:4) but I would note that their frequencies ($1/\tau^2$ and $1/\tau^4$, respectively) are relatively low; of these two, the vertex star with higher frequency, the skinny box SB, involves only three tiles (and thus would be expected to have a higher rate of formation). Thus the vertex stars having a low fat:thin rhomb ratio that *are observed* are consistent with explanation (ii).

Appendix D

Capillary force based gears

One day, Len and I were walking down Blum Walk discussing patterns that one could make from hydrophilic and hydrophobic patches. Len asked, "What if you made alternating patches around the perimeter of a disk, would it spin?" "I don't think so," I answered, "but if you made two of them they would mesh like gears but they wouldn't have to touch." One could make non-contacting gears based on capillary action!

Two gears (1.2 cm diameter) like that in Figure D.1 were constructed from 2 mm thick clear polycarbonate (Lexan) sheet; one gear had a steel rod (1 mm diameter, 9 mm long) glued into a slot on the surface (the driver), and the other had the same size brass rod glued in a similar slot (the driven gear) so that both gears had the same density. Twenty patches of alternating hydrophilic and hydrophobic character were added to the tiles by methods described in Chapter 3. The gears were floated at the interface of hexadecane and a sodium metatungstate solution in a glass dish, and the dish was suspended so that the interface was $\sim$ 4 cm over a magnetic stir bar anchored to a freely revolving disk. In seconds the driver gear localized itself over the magnet (and was pulled down $\sim$ 1-2 mm by the magnet). Placed within $\sim$ 2 mm of the driver the driven gear was attracted to it — the

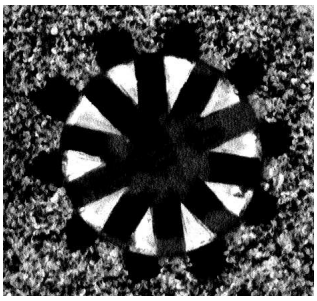


Figure D.1: Gear in hydrophobic sand. A capillary force gear with 20 "teeth": 10 hydrophilic patches and 10 hydrophobic patches. Light wedges on the gear indicate hydrophilic patches — sand moves away from these patches forming dark lobes.

driven gear moved until it contacted the driver gear with its hydrophilic and hydrophobic patches aligned. This process is interesting since the driven gear accelerates to the driver, rotating to a correct alignment as it accelerates, and stops abruptly "snapping" into an alignment with meshed teeth. Occasionally, if pushed towards the driver gear with almost perfect misalignment, the driven gear is repelled and bounces back.

Spinning the magnet caused the driver gear to spin at the same rate. At driver speeds that were too low the driven gear would revolve with the driver gear, its center tracing out a circle about the axis of the rotating magnet. As the speed of the driver increased, the drag on the driven gear encouraged it to rotate on its axis, in a direction opposite to the driver gear, maintaining tooth alignment in the process. As the speed of the driver increased further, the driven gear rotated faster, and the tendency for it to revolve around the driver gear became less. In this sense, the gears seemed to work best if the driver was driven at approximately 12–20 rpm — at these rates the driven gear seemed to rotate once for every revolution that it made about the center of the magnet; for example when the driver rotated at ~ 20 rpm the driven gear rotated at ~ 10 rpm and revolved at ~ 10

rpm. At driver speeds that were too great (above about 20 rpm), the driven gear would detach from the driver gear, "fly off" a few centimeters, and had to be gently pushed back towards the driver.

The gears switched often between *sticking* (one pair of teeth remained aligned and the driven gear just revolved) and *smooth operation* (the teeth remained in good alignment, passed over each other freely, and the driven gear mainly rotated). That is, the fraction of the driver's motion that was converted into rotation of the driven gear versus revolution varied widely even over the course of a few seconds. We attribute this partially to the fact that the magnet was manually rotated (and thus its angular velocity was at times erratic) and because we believe that the teeth were not made completely uniformly, some being slightly larger than others and more prone to stick, perhaps. Also some teeth may have been rough — the cyanoacrylate/silica matrix on the hydrophilic patches has small bumps. This is supported by our observation that the gears seemed to operate more smoothly and the driven gear seemed to detach less frequently (for a given driver rpm) when the driver was rotated in a counterclockwise direction. Perfectly symmetric gears should not have this behavior. It is encouraging that, with the driver rotating counterclockwise, we observed 6-7 rotations of the driven gear (with respect to teeth on the driver gear) without the driven gear sticking completely or detaching (this took 11-12 rotations of the driver gear over 29 seconds).

We didn't take this project very far. There are, so far, three interesting directions to pursue it:

1. Basic physics of the gears. We would like to know how good these gears are — at what distance they can operate, how efficiently they can transfer energy (and how

much they dissipate to the fluids), and at what speeds they *slip* (that is, the teeth lose alignment). Studying the gears when they are free to move at the interface is difficult and doesn't inform us much about how they would act in a mechanical system. The obvious experiment is to put a driver and a driven gear on "shafts" — poles oriented vertically through the interface on which they can freely rise up and down. Then, observe their behavior as a function of the separation and driving rate. The drag force exerted by the fluid, as a function of gear rpm, can be found by accelerating the driver gear alone, to a sufficient rpm, removing the rotating magnet underneath quickly (say it is an electromagnet) and watching the driver gear decelerate under viscous drag from the fluid. This information, combined with the two gear experiment should allow one to measure the efficiency of the gears.

Also important is how the behavior of the gears scales with the size of the gears. Perhaps these gears will work best in micromechanical systems. They might be desirable in such systems since they should exhibit very little wear.

2. Self-organizing patterns from gears. Imagine an entire flotilla of driver and driven gears place over a large, uniform, magnetic field that *may* be rotated and gently shaken. Consider first what may happen if the field isn't rotated. The driver gears would align with the field; further, if the field was sufficiently uniform, they would have little tendency to clump. (In our experiments, the field was sufficiently localized that the driver was pinned directly over the magnet. With such a setup all the drivers would clump near the magnet.) Then, shaken gently, the drivers and driven gears would move to minimize the interfacial energy. With suitable tooth spacing

(in principle, multiples of 6 teeth/gear should work; other spacings may work less well) we expect that the gears might collapse into a hexagonally packed crystalline arrangement where the identity of a gear at any given site, driver or driven, is random. This a "ground state" (Figure D.2A) that we would like to observe.

Now imagine continuing the gentle shaking but rotating the magnetic field slowly beneath them. The driver gears would then be forced to rotate, say clockwise. Driven gears, in contact or close to driver gears, would be forced to rotate counterclockwise. These motions are incompatible with the hexagonally packed arrangement — there would always exist two neighbors in a trio of gears which are trying to rotate in the same direction and whose teeth cannot align. Thus we believe that the gears might assume a square packing, a checkerboard of alternating driver and driven gears. This an "excited state" (Figure D.2A) that we would like to observe. A simple version of this experiment would be to observe the behaviour of four gears, two drivers and two driven, and observe their transition between a rhomboidal ground state and an excited square state (Figure D.2B).

These experiments would be interesting because they would demonstrate how self-assembly, in combination with a little coordinated motion, can organize interesting patterns.

3. A pump based on the gears. Such a pump seems interesting because it seems like it could be easily incorporated into a microfabricated fluid handling system. Like the capillary gears it would exhibit very little wear.

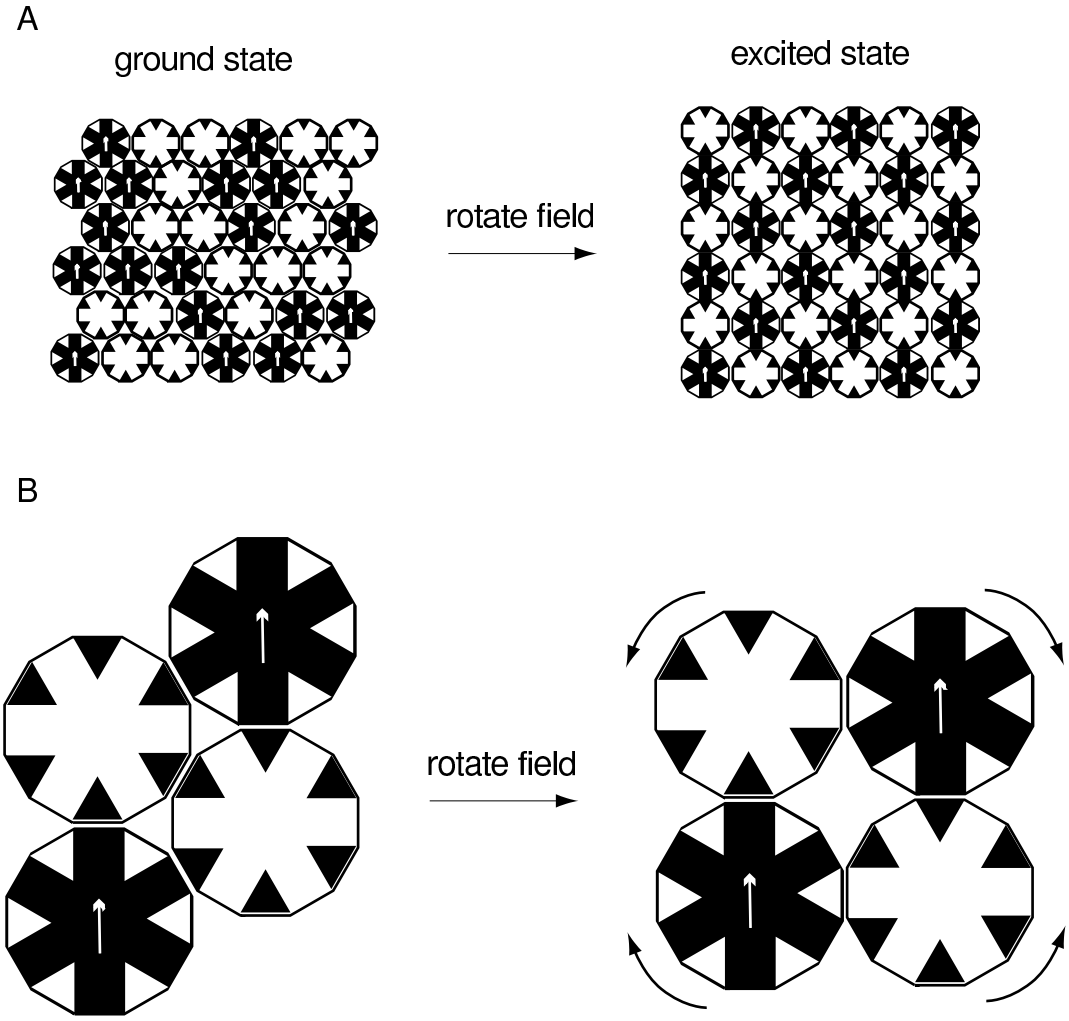


Figure D.2: Hypothetical self-assembly of capillary force gears. Largely black gears with arrows are driver gears; largely white gears are driven gears. We intend that where the edge of a gear is white it is hydrophilic and where it is black it is hydrophobic. Gears are drawn as octagons but are intended to be disks. (A) We predict that gears shaken gently over a large fixed magnetic field assume a hexagonally packed ground state where each site is occupied by a driver or driven gear at random. We predict that if the field is shaken and rotated slowly, the gears will assume a square-packed excited state with driver and driven gears arranged in a checkerboard pattern. (B) We base our prediction, in part, on the fact that three gears arranged in a triangle cannot rotate in a mutually consistent manner. Thus we expect the rhomboidal arrangement at left to transform to the square arrangement at right if the magnetic field is rotated. Driver gears rotate clockwise and driven gears rotate counterclockwise.

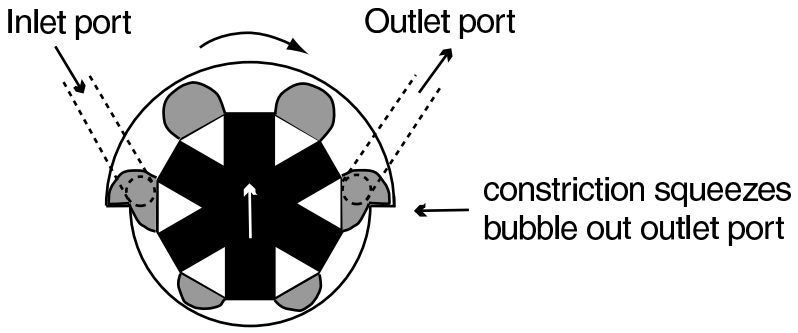


Figure D.3: Plan for a possible capillary-forced based pump. Gray blobs indicate water. The black contour around the gear and blobs of water is a hydrophobic housing. It widens at the input port and narrows at the output port. A curved arrow shows the direction of motion of the gear.

John McCaskill suggested the idea and we developed it a little together: put a single gear in a disk shaped housing with two holes or ports (geometrically undifferentiated) on the face of the disk shaped housing 180 degrees apart. Further, make the pattern of hydrophilic and hydrophobic patches on the housing asymmetric so that the ports are differentiated into an inlet port and an outlet port. Our intent is that the hydrophilic patches are wet with blobs of water and the rest of the space is taken up by a thin oil. When the gear is spun, the idea is that it may pick up water on its hydrophilic patches at the inlet port, and that, as it passes the outlet port it passes a hydrophilic patch that effectively squeezes the water out. It is not clear that such a geometry of hydrophilic and hydrophobic patches exists.

An alternative design is to use a geometric constriction of hydrophobic material at the output port, and a geometric widening at the input port. This design is shown in Figure D.3.