2009

Applications of the theory of computation to nanoscale self-assembly

David Doty
Iowa State University

Applications of the theory of computation to nanoscale self-assembly

by

David Samuel Doty

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:
Jack H. Lutz, Co-major Professor
James I. Lathrop, Co-major Professor
Pavan Aduri
John Mayfield
Elvira Mayordomo

Iowa State University

Ames, Iowa

2009

Table of Contents

List of Figures

Acknowledgements

The National Science Foundation supported this thesis through grants 0652569, 0728806, and CCF:0430807.

I thank foremost my advisors Jack Lutz and James Lathrop. Their guidance gradually transformed me from a good student into a good researcher and teacher. Our department chair Carl Chang gave me the unique opportunity to instruct classes, which has been among the most enjoyable experiences of graduate school. Pavan Aduri treated me like one of his own students and allowed me to treat him like an advisor. Elvira Mayordomo, Philippe Moser, and Xiaoyang Gu taught me how to do research as a sprint, and Satyadev Nandakumar, Matt Patitz, and Scott Summers taught me how to do research as a marathon (and to conserve oxygen when appropriate). The latter three have been especially instructive in the social nuances of research. My former advisors John Mayfield and Dan Ashlock introduced me to mathematical theory, and what it has to say about the world we inhabit, in a way that opened my eyes and changed my view of the purpose of mathematics. Dean Adams rightfully torpedoed my original preliminary research proposal, an act that ultimately set me on my present, more successful, course of research. Science needs more people like Dean, who have the courage to say "no" when under pressure to say "yes". My family has supported me emotionally and at times financially. They did not always understand why I chose the path I did, but they stood behind me nevertheless.

Finally, my entire career is owed to my wife Julie, in ways that can never be repaid. Few appreciate the unique emotional trauma that accompanies graduate school, but Julie does. Her loving support is the reason I finished my Ph.D., and it is the reason I continue my life as a scientist.

Abstract

This thesis applies the theory of computing to the theory of nanoscale self-assembly, to explore the ability – and under certain conditions, the inability – of molecules to automatically arrange themselves in computationally sophisticated ways. In particular, we investigate a model of molecular self-assembly known as the abstract Tile Assembly Model (aTAM), in which different types of square "tiles" represent molecules that, through the interaction of highly specific binding sites on their four sides, can automatically assemble into larger and more elaborate structures.

We investigate the possibility of using the inherent randomness of sampling different tiles in a well-mixed solution to drive selection of random numbers from a finite set, and explore the tradeoff between the uniformity of the imposed distribution and the size of structures necessary to process the sampled tiles.

We then show that the inherent randomness of the competition of different types of molecules for binding can be exploited in a different way. By adjusting the relative concentrations of tiles, the structure assembled by a tile set is shown to be programmable to a high precision, in the following sense. There is a single tile set that can be made to assemble a square of arbitrary width with high probability, by setting the concentrations of the tiles appropriately, so that all the information about the square's width is "learned" from the concentrations by sampling the tiles.

Based on these constructions, and those of other researchers, which have been completely implemented in a simulated environment, we design a high-level domain-specific "visual language" for implementing complex constructions in the aTAM. This language frees the implementer of an aTAM construction from many low-level and tedious details of programming

and, together with a visual software tool that directly implements the basic operations of the language, frees the implementer from almost any programming at all.

Finally, after showing these positive results, we turn our attention to negative results and investigate inherent limitations in the aTAM at "temperature 1", meaning roughly that all bonds in the system have sufficient strength to permanently attach tiles without help from other bonds (i.e., the temperature is too low to "shake off" any tiles, even those connected by a single bond). Specifically, we show that at temperature 1, a wide class of deterministic tile sets (those satisfying a natural condition known as "pumpability") form only the most computationally simple structures (specifically, semilinear sets of integer coordinates, equivalently those sets definable in Presburger arithmetic), and in particular are strictly less powerful than the computationally universal temperature 2 tile assembly model. We leave as an open question whether all deterministic temperature 1 tile sets are in fact pumpable.

Chapter 1. Overview

1.1 Introduction

1.1.1 Why is computation beneficial?

What is it about algorithmic computation, and our ability to implement it with electronic components, that makes it so useful? The obvious answer may be *speed*. For instance, programmers often find themselves, when working on a large programming project, writing a build script, which quickly handles all the tedious steps that must be taken to create a program from its source code: compiling, linking libraries, packaging into a single executable file along with images and other data files, perhaps uploading the whole thing to a web server.

Is speed the only reason to do this? Even if the build script took twice as long as doing all this by hand, programmers would still write it. Why? There is another reason: *correctness*. The build script does not make the mistakes that every programmer eventually makes.

But I argue that we are still misdiagnosing the disorder that the build script cures. If my build script worked at half the speed I do, and one in every ten times failed to build the program, forcing me to re-run it, I would continue in my stubborn ways and use the script in lieu of building the program manually. What makes that build script so useful, if not speed or correctness? I contend that *automation* is the script's *raison d'être*; since my time is worth much more than my computer's, if the script can free me of 5 minutes to think about fixing bugs or designing the next module, that script can be remarkably inefficient, and even require the occasional restart, and I will thank Turing for the privilege of using it.

1.1.2 Why is computation at molecular scales more than just beneficial?

While the previous example shows that an automatic build script gives a programmer extra time to think about design (or to shop online or to daydream), the script is not, strictly speaking, *necessary*. Nanoscience provides a novel justification for studying computation, and for studying the kinds of computation that can happen at the nanoscale. This reason is that many of the traditional forms of manual control are simply *not possible* at small scales. We will automate the building of molecular structures not because it is faster than building such structures by hand. Rather, our hands, and the machines that they operate, are too large to manipulate individual molecules, so we simply have no other choice. We must learn to program molecules to manipulate themselves.

1.2 Nanoscale Self-Assembly

1.2.1 Practice

Ned Seeman [35] proposed a novel approach to the problem of protein crystallization, the first – and often the most difficult – step of determining the structure of a protein through the technique of x-ray crystallography. He designed DNA sequences with the property that their natural tendency was to form what is known as a *double-crossover molecule*: two single strands of DNA, each of which is bound to complementary strands of DNA to form regions of double-strandedness, which at two points form crossovers, meaning the complementary strands are exchanged between the single strands. Thus the structure really consists of four individual strands, but we single out two as "primary" and interpret the other two to "cross over" from one primary strand to the other primary strand. Given the correct length and temperature parameters, the double crossover molecule forms into something we may logically visualize as a cross, as shown in Figure 1.1.

The single strands protruding from each of the four sides are known as *sticky ends*, because each will stick to its Watson-Crick complementary strand: adenine (A) to thymine (T) and cytosine (C) to guanine (G). Finally, observe that the north sticky end is complementary to

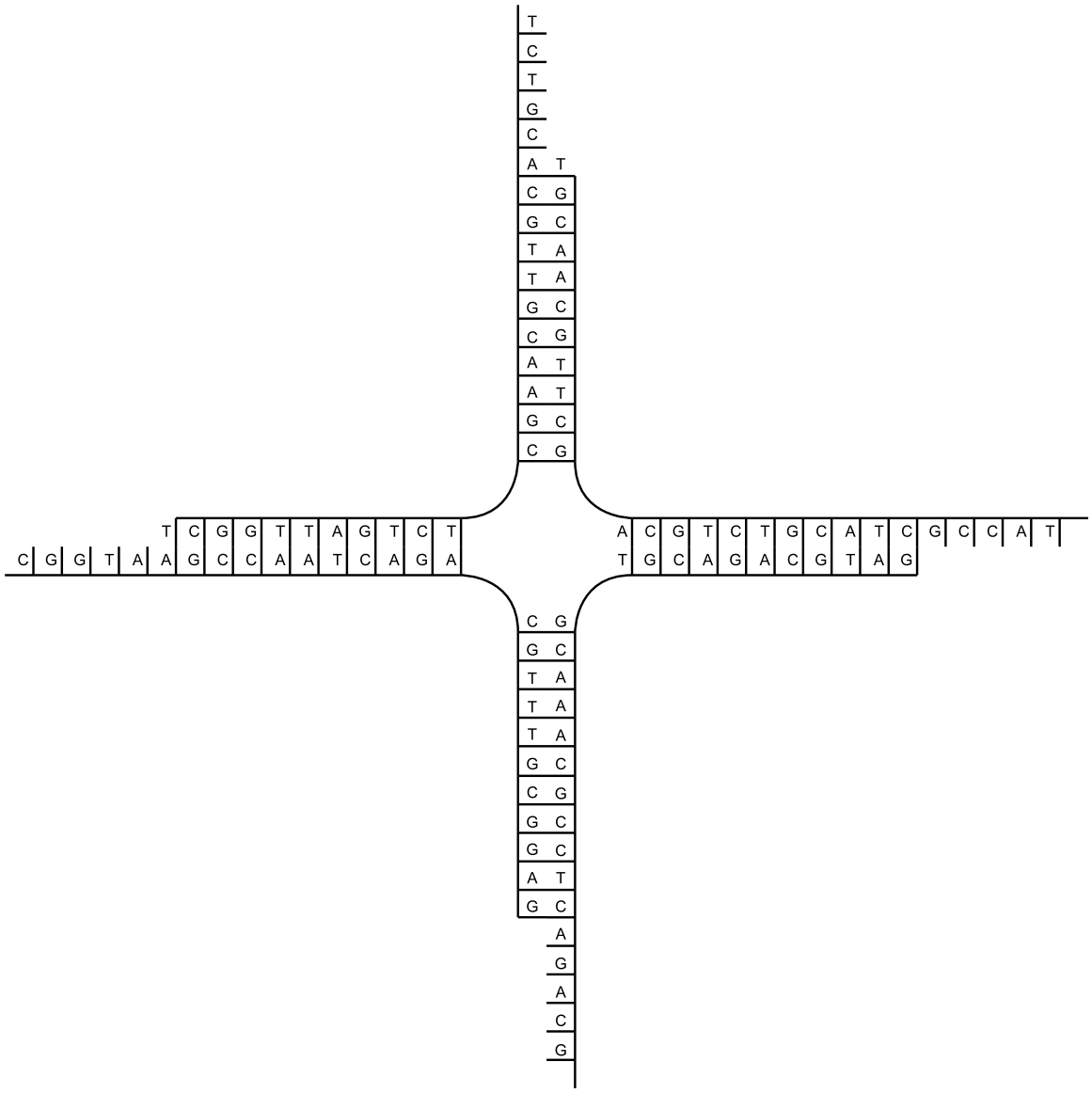


Figure 1.1: Logical depiction of a double-crossover molecule. Note the sticky ends coming from each of the four sides, which give the double-crossover molecule its binding constraints.

the south, and the west to the east, so that many copies of the molecule of Figure 1.1, mixed in solution, will form the crystal lattice depicted in Figure 1.2. Seeman hoped to simplify the crystallization step of crystallography by designing each double-crossover molecule to hold exactly one protein in place, thus arranging all proteins in a crystal lattice, oriented in the same direction, which is necessary for determining the protein's structure through crystallography.

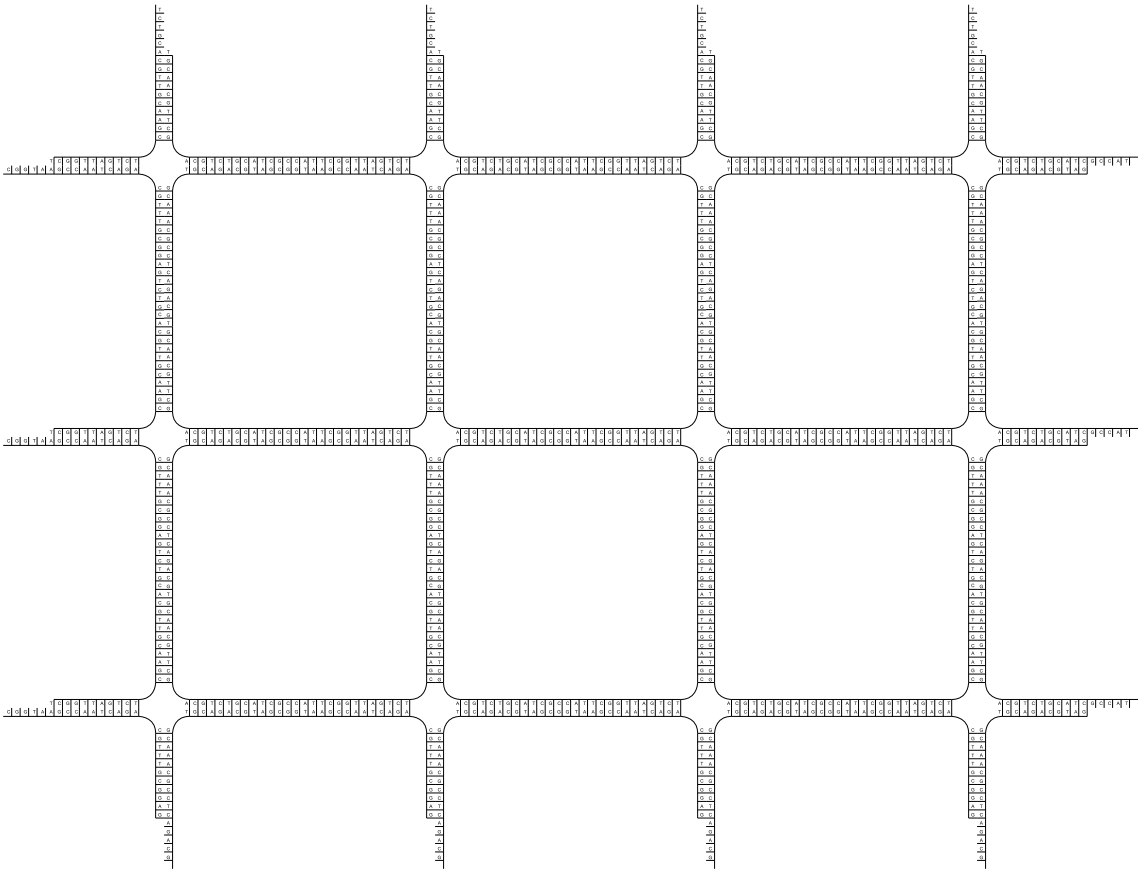


Figure 1.2: Lattice assembled by many copies of the double-crossover molecule of Figure 1.1.

This biochemical application remains experimental, but Erik Winfree, a theoretical computer scientist, noticed something extraordinary. If there were more than one type of double-crossover molecule, and if the sticky ends were set up to allow different types of molecules to bind to each other, then much more sophisticated structures could be created; one example is shown in Figure 1.3. In fact, what emerges from this simple generalization is a theory quite

similar to the computationally rich model of *tiling* studied by the logician Hao Wang [43, 44].

Using techniques borrowed from Seeman, Winfree and others have created self-assembled structures from DNA tiles in the laboratory [3, 9, 23–25, 33, 48]. These experiments provide the physical basis for the theoretical questions explored in this thesis.

1.2.2 Theory

In this section we give a semi-formal introduction to Winfree's mathematical model of self-assembling DNA tiles, known as the *abstract Tile Assembly Model (aTAM)*. Chapter 2 provides a complete formal specification of the aTAM. These concepts were introduced in [45], and the notions of weak and strict self-assembly were first studied in their present form in [22].

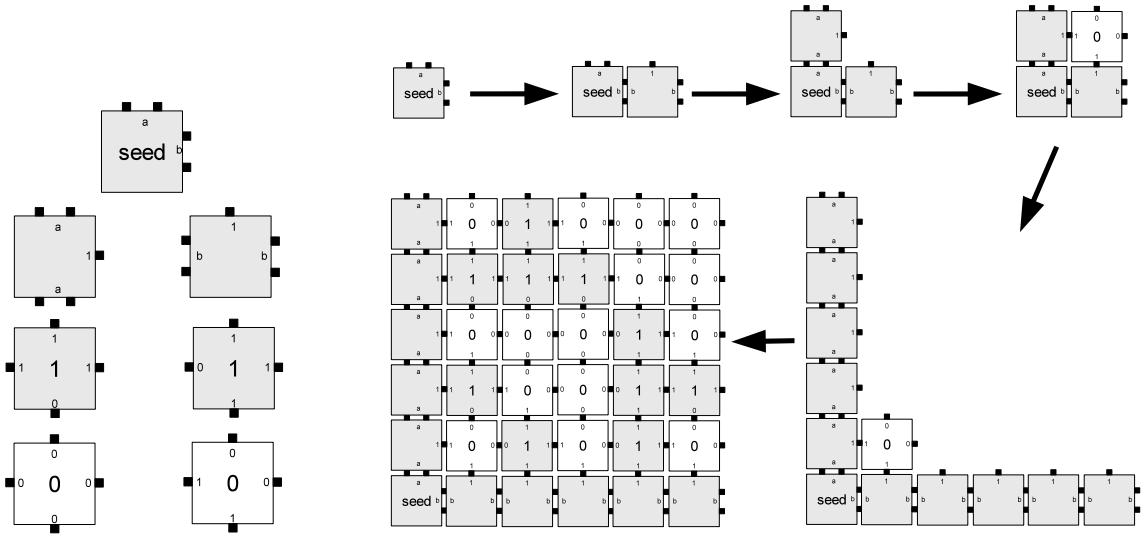
The double-crossover molecules discussed in Section 1.2.1 are modeled as unit squares called *tiles* that cannot be rotated (in reality, of course they rotate, but the binding rules can be set such that assembly will only occur between squares that are oriented in the same un-rotated direction). Each side of a tile has a *glue*, consisting of a *label* (modeled as a finite string for the sake of drawing figures) and a non-negative integer *strength*, usually 0, 1, or 2. The four glues of each side define a *tile type*, and two tiles of the same type are functionally identical. There are a finite number of tile types, and an infinite supply of copies of each tile type. Assembly begins with a single copy of the specially designated *seed* tile type, placed at the origin of the integer lattice $\mathbb{Z}^2$. The fundamental transition rule is this: given the assembly that has grown thus far (beginning with the assembly consisting of just the seed), a tile may attach to the assembly at a given point if the sum, over all sides of the tile whose glue label matches that of the glue to which it is adjacent, of the strengths of those glues, is at least the system's ambient *temperature* (usually set to be 2, although sometimes 1). Any empty position adjacent to an assembly at which some tile could bind is known as a *frontier location*. The tile set, when taken together with the seed tile (or sometimes, a multi-tile *seed assembly*), the temperature value, and whatever other information is necessary to specify the environment (in generalizations of the model), is referred to as a *tile assembly system (TAS)*.

Stated plainly, the transition rule for temperature-2 assembly implies that a tile may attach

to an assembly if it may be placed so that two of its strength-1 glues match the glues of the abutting sides of two adjacent tiles in the assembly, or if one of its strength-2 glues matches the glue of the abutting side of an adjacent tile in the assembly. The former case is a phenomenon informally termed *cooperation*, in the sense that two different tiles already existing in the assembly are required to *both* match sides of the new tile before it can be placed. The effect is that the newly attached tile "computes a function" of two inputs, where two different tiles in the existing assembly provide the input values, and the output is advertised on the non-input sides of the newly-attached tile (perhaps propagating a different output in two different directions). This cooperation is key to many advanced constructions in the aTAM. Note that, unlike the related model of Wang tiling [43, 44], a tile may attach even if some of its adjacent glues do not match those of the assembly, so long as sufficient binding strength is received from the sides that match. In other words, there are no negative glue strengths. Note that this implies monotone growth; once a tile attaches to the assembly, it never detaches.

Figure 1.3 shows an example of a tile set, due to Winfree [45], and the first few stages of growth of the unique assembly that it forms. The entire first quadrant is filled by tiles in the limit, and the pattern "painted" is alternately known as the *discrete Sierpinski triangle* or *Pascal's triangle modulo 2*.

There are multiple senses in which a TAS may be nondeterministic. Nearly all nontrivial TAS's have the property that they may grow into an assembly with more than one frontier location. The frontier location for the next attachment is selected nondeterministically, with the selection scheme being a parameter of the theory. In all of our constructions, any selection scheme will suffice, so long as it is *fair*, meaning that any frontier location will eventually have a tile placed there (none of the frontier locations are "starved"). Such a TAS may still be deterministic in the sense that all fair assembly sequences eventually result in the same assembly. The more significant sense in which tile assembly may be nondeterministic is that there is more than one *terminal* assembly, terminal meaning an assembly with an empty frontier. This is easily seen to be equivalent to the condition that in every possible sequence of tile additions, for each frontier location of each intermediate nonterminal assembly, there is



(a) Example tile set that assembles the discrete Sierpinski triangle.

(b) Possible first few stages of assembly.

Figure 1.3: Example tile set that weakly assembles the discrete Sierpinski triangle, and some examples of potential initial stages of assembly. Strength-2 glues are illustrated with two lines between tiles, strength-1 glues are illustrated with a single line between tiles, and strength-0 glues (i.e., the absence of a glue) have no lines. Using the analogy that cooperation between strength-1 bonds is like computing a function of two inputs, the input each of the non-border tiles computes is the XOR of the bits on the south and west, the output of which appears on the north and east side.

only one tile that could be placed *at that location*. A TAS that is deterministic in this sense is alternately known as *directed*, *confluent*, or is said to *produce a unique assembly*.

The term *assembly* refers not only to what positions have a tile, but also what tile is placed at each position. Two different assemblies may represent the same shape, in the sense that they have tiles placed at the same set of positions but have at least one position at which they have placed different types of tiles. If the TAS $\mathcal{T}$ is guaranteed to produce a terminal assembly with tiles at exactly the positions in some set $X \subseteq \mathbb{Z}^2$, we say X *strictly self-assembles* in $\mathcal{T}$ (even if it is not guaranteed *which* tile will be placed at each position in X).[1] Furthermore, we may relax the definition of "shape" somewhat, defining *weak self-assembly*. Instead of asking what locations have a tile in an assembly, we paint some of the tile types "black" (i.e., designate a subset of the tile types that will represent the shape), and define the shape of the assembly to be those locations occupied by a black tile. For instance, by defining all but the two tile types labeled in the center with '0' in Figure 1.3 to be black, the TAS weakly self-assembles the discrete Sierpinski triangle, or equivalently the set $\left\{ (x, y) \in \mathbb{N}^2 \;\middle|\; \binom{x+y}{y} \text{ is odd} \right\}$. The same TAS strictly self-assembles the entire first quadrant. The results of weak self-assembly can be measured, for instance, by placing hairpin loops on the black tile types, which can then be scanned by atomic force microscopy [33].

In fact, the aTAM is capable of much more. Winfree [45] showed that every algorithm can be simulated by a TAS, in the sense that the TAS forms an assembly representing the space-time configuration history of a cellular automaton implementing the algorithm.

1.3 Applications of the Theory of Computation to Nanoscale Self-Assembly

This section informally outlines the original contributions of this thesis.

[1] Although it is difficult to imagine a counterexample, it is an open problem to prove that if a set strictly self-assembles in some TAS, then it strictly self-assembles in some directed TAS. The same problem is currently open for weak self-assembly.

1.3.1 Random Number Selection in Self-Assembly

In Chapter 3, we investigate methods for exploiting nondeterminism inherent within the aTAM in order to generate uniform random numbers. Namely, given an integer range $\{0, \ldots, n-1\}$, we exhibit methods for randomly selecting a number within that range. We present three constructions exhibiting a trade-off between space requirements and closeness to uniformity.

The first selector selects a random number with probability $\Theta(1/n)$ using $O(\log^2 n)$ tiles. The second selector takes a user-specified parameter that guarantees the probabilities are arbitrarily close to uniform, at the cost of additional space. The third selector selects a random number with exact uniformity (i.e., probability exactly $1/n$), and uses no more space than the first selector with high probability, but uses potentially unbounded space.

The final construction is a self-assembly implementation of von Neumann's *rejection method* [20, 42], which is equivalent to the algorithm used to select random numbers in programming language libraries such as the Java's standard library. In a programming language, a "ZPP" algorithm that almost certainly takes a small amount of time, but may rarely use more time, is not a problem except perhaps in critical real-time systems. However, in a tile assembly system, consisting of possibly billions of tiles, the presence of even a single region that uses too much space will destroy the correctness of the entire assembly. In such situations, it may be preferable to absolutely guarantee that space bounds are obeyed at the cost of a slight deviation from uniformity.

1.3.2 Randomized Self-Assembly for Exact Shapes

In Chapter 4, we prove a result in a generalization of the aTAM known as *tile concentration programming*. In this model, each tile type is assumed to be present in solution in a certain experimenter-controlled *concentration*, which, if more than one tile type compete nondeterministically to bind to a single frontier location, determines the probability that each will be the one to bind.

We show that a constant-size tile assembly system can be programmed through tile concentration programming to build an $n \times n$ square with high probability, for any sufficiently

large n. This answers an open question of Kao and Schweller [19], who showed how to build an *approximately* $n \times n$ square using tile concentration programming, and asked whether the approximation could be made *exact* with high probability. This technique can be modified to answer another question of Kao and Schweller [19], by showing that a constant-size tile assembly system can be programmed through tile concentrations to assemble arbitrary finite *scaled shapes* (a concept introduced for the aTAM in [38]), which are shapes modified by replacing each point with a $c \times c$ block of points, for some integer c.

1.3.3 A Domain-Specific Language for Programming in the Tile Assembly Model

The tile assembly systems of Chapters 3 and 4 were not produced by hand; rather they were output by a program that reused code for groups of related tile types.

In Chapter 5, we introduce a domain-specific language (DSL) for creating sets of tile types for simulations of the aTAM. The language defines objects known as tile templates, which represent related groups of tiles, and a small number of basic operations on tile templates that help to eliminate the error-prone drudgery of enumerating such tile types manually or with low-level constructs of general-purpose programming languages. The language is implemented as a class library in Python (a so-called *internal DSL*), but is presented independently of Python or object-oriented programming, with emphasis on support for a visual editing tool for creating large sets of complex tile types.

1.3.4 Limitations of Self-Assembly at Temperature One

In Chapter 6, we explore what sort of assembly and computation are possible in the aTAM when the temperature is set to 1. This is equivalent to eliminating *cooperativity* from the model; i.e., changing the rules so that a tile may bind to an assembly if *any single one* of its glues match those of an exposed side of a tile in the assembly. Our goal is to show that cooperativity is essential for advanced computation in the aTAM, by showing that temperature 1 tile assembly is strictly weaker than assembly at temperature 2.

We prove that if a set $X \subseteq \mathbb{Z}^2$ weakly self-assembles at temperature 1 in a deterministic

(Winfree) tile assembly system satisfying a natural condition known as *pumpability*, then X is a finite union of linear sets. A set is *linear* if it is expressible as $\left\{ \vec{b} + n \cdot \vec{u} + m \cdot \vec{v} \;\middle|\; n, m \in \mathbb{N} \right\}$ for some fixed vectors $\vec{b}, \vec{u}, \vec{v} \in \mathbb{Z}^2$; i.e., it consists of the vertices of an infinite array of parallelograms packed next to each other. A finite union of linear sets is also known as a *semilinear* set. Semilinear sets are computationally very simple. They have been shown, for example, to be equivalent to those sets definable in Presburger arithmetic [29, 40], the first-order theory of natural numbers with only the addition operation (but lacking multiplication). For completeness, we give a self-contained proof (Observation 6.3.2) that the projection of any semilinear set onto the x-axis, when expressed in unary, is a regular language, to emphasize the computational simplicity of semilinear sets.

Informally, a *pumpable* temperature 1 tile assembly system is one with the property that every sufficiently long path of tiles in an assembly of this system contains a segment in which the same tile type repeats (a condition clearly implied by the pigeonhole principle), and that furthermore, the subpath between these two occurrences can be repeated indefinitely ("pumped") along the same direction as the first occurrence of the segment, without "colliding" with a previous portion of the path. We give a counterexample in Section 6.2 (Figure 6.1) of a path in which the same tile type appears twice, yet the segment between the appearances cannot be pumped without eventually resulting in a collision that prevents additional pumping. The hypothesis of pumpability states (roughly) that in every sufficiently long path, despite the presence of some repeating tiles that cannot be pumped, *there exists* a segment in which the same tile type repeats that *can* be pumped.

This shows that only the most simple of infinite shapes and patterns can be constructed using pumpable temperature 1 tile assembly systems, and gives evidence for the thesis that temperature 2 or higher is required to carry out general-purpose computation in a tile assembly system. We employ this result to show that, unlike the case of temperature 2 self-assembly, no discrete self-similar fractal weakly self-assembles at temperature 1 in a pumpable tile assembly system. Finally, we show that general-purpose computation *is* possible at temperature 1 if negative glue strengths are allowed in the tile assembly model.

It remains open whether every deterministic temperature 1 tile assembly system is pumpable; if so, this would settle the question completely, showing that deterministic temperature 1 tile assembly systems assemble only the most computationally simple sets.

Chapter 2. The Abstract Tile Assembly Model

We work in the 2-dimensional discrete space $\mathbb{Z}^2$. Define the set $U_2 = \{(0,1), (1,0), (0,-1), (-1,0)\}$ to be the set of all *unit vectors*, i.e., vectors of length 1 in $\mathbb{Z}^2$. We write $[X]^2$ for the set of all 2-element subsets of a set X. All *graphs* here are undirected graphs, i.e., ordered pairs $G = (V, E)$, where V is the set of *vertices* and $E \subseteq [V]^2$ is the set of *edges*.

Intuitively, a tile type t is a unit square that can be translated, but not rotated, having a well-defined "side $\vec{u}$" for each $\vec{u} \in U_2$. Each side $\vec{u}$ of t has a "glue" with "label" $\text{lab}_t(\vec{u})$ – a string over some fixed alphabet Σ – and "strength" $\text{str}_t(\vec{u})$ – a nonnegative integer – specified by its type t. Two tiles t and t' that are placed at the points $\vec{a}$ and $\vec{a} + \vec{u}$ respectively, *bind* with *strength* $\text{str}_t(\vec{u})$ if and only if $(\text{lab}_t(\vec{u}), \text{str}_t(\vec{u})) = (\text{lab}_{t'}(-\vec{u}), \text{str}_{t'}(-\vec{u}))$.

Given a set T of tile types, an *assembly* is a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$, with points $\vec{x} \in \mathbb{Z}^2$ at which $\alpha(\vec{x})$ is undefined interpreted to be empty space, so that dom α is the set of points with tiles. α is *finite* if $|\text{dom } \alpha|$ is finite. For assemblies α and α', we say that α is a *subconfiguration* of α', and write $\alpha \sqsubseteq \alpha'$, if dom $\alpha \subseteq$ dom α' and $\alpha(\vec{x}) = \alpha'(\vec{x})$ for all $x \in$ dom α.

Let α be an assembly and $B \subseteq \mathbb{Z}^2$. α *restricted to* B, written as $\alpha \upharpoonright B$, is the unique assembly satisfying $(\alpha \upharpoonright B) \sqsubseteq \alpha$, and dom $(\alpha \upharpoonright B) = B$. If π is a sequence over $\mathbb{Z}^2$ (such as a path), then we write $\alpha \upharpoonright \pi$ to mean α restricted to the set of points in π. If $A \subseteq$ dom α, we write $\alpha \setminus A = \alpha \upharpoonright (\text{dom } \alpha - A)$. If $\vec{0} \neq \vec{v} \in \mathbb{Z}^2$, then the *translation of* α *by* $\vec{v}$ is defined as the assembly $(\alpha + \vec{v})$ satisfying, for all $\vec{a} \in \mathbb{Z}^2$,

$$(\alpha + \vec{v})(\vec{a}) = \begin{cases} \alpha(\vec{a}) & \text{if } \vec{a} - \vec{v} \in \text{dom } \alpha \\ \uparrow & \text{otherwise.} \end{cases}$$

A *grid graph* is a graph $G = (V, E)$ in which $V \subseteq \mathbb{Z}^2$ and every edge $\{\vec{a}, \vec{b}\} \in E$ has the property that $\vec{a} - \vec{b} \in U_2$. The *binding graph of* an assembly α is the grid graph $G_\alpha = (V, E)$, where $V = \text{dom } \alpha$, and $\{\vec{m}, \vec{n}\} \in E$ if and only if (1) $\vec{m} - \vec{n} \in U_2$, (2) $\text{lab}_{\alpha(\vec{m})}(\vec{n} - \vec{m}) = \text{lab}_{\alpha(\vec{n})}(\vec{m} - \vec{n})$, and (3) $\text{str}_{\alpha(\vec{m})}(\vec{n} - \vec{m}) > 0$. An assembly is *$\tau$-stable*, where $\tau \in \mathbb{N}$, if it cannot be broken up into smaller assemblies without breaking bonds of total strength at least τ (i.e., if every cut of G_α cuts edges, the sum of whose strengths is at least τ).

Self-assembly begins with a *seed assembly* σ (typically assumed to be finite and τ-stable) and proceeds asynchronously and nondeterministically, with tiles adsorbing one at a time to the existing assembly in any manner that preserves stability at all times.

A *tile assembly system* (*TAS*) is an ordered triple $\mathcal{T} = (T, \sigma, \tau)$, where T is a finite set of tile types, σ is a seed assembly with finite domain, and τ is the temperature. An *assembly sequence* in a TAS $\mathcal{T} = (T, \sigma, \tau)$ is a (possibly infinite) sequence $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ of assemblies in which $\alpha_0 = \sigma$ and each α_{i+1} is obtained from α_i by the "τ-stable" addition of a single tile. The *result* of an assembly sequence $\vec{\alpha}$ is the unique assembly $\text{res}(\vec{\alpha})$ satisfying $\text{dom res}(\vec{\alpha}) = \bigcup_{0 \leq i < k} \text{dom } \alpha_i$ and, for each $0 \leq i < k$, $\alpha_i \sqsubseteq \text{res}(\vec{\alpha})$.

Note that, if $\mathcal{T} = (T, \sigma, 1)$, then there is a finite path from the seed to a point $\vec{x} \in \text{dom } \alpha$, denoted as $\pi_{\vec{0},\vec{x}}$ if and only if there is an assembly sequence $\vec{\alpha}$ satisfying $\text{res}(\vec{\alpha}) = \alpha \upharpoonright \pi_{\vec{0},\vec{x}}$.

We write $\mathcal{A}[\mathcal{T}]$ for the *set of all producible assemblies of* $\mathcal{T}$. An assembly α is *terminal*, and we write $\alpha \in \mathcal{A}_\square[\mathcal{T}]$, if no tile can be stably added to it. We write $\mathcal{A}_\square[\mathcal{T}]$ for the *set of all terminal assemblies of* $\mathcal{T}$. A TAS $\mathcal{T}$ is *directed*, or *produces a unique assembly*, if it has exactly one terminal assembly i.e., $|\mathcal{A}_\square[\mathcal{T}]| = 1$. The reader is cautioned that the term "directed" has also been used for a different, more specialized notion in self-assembly [2]. We interpret "directed" to mean "deterministic", though there are multiple senses in which a TAS may be deterministic or nondeterministic.

A set $X \subseteq \mathbb{Z}^2$ *weakly self-assembles* if there exists a TAS $\mathcal{T} = (T, \sigma, 1)$ and a set $B \subseteq T$ (B constitutes the "black" tiles) such that $\alpha^{-1}(B) = X$ holds for every assembly $\alpha \in \mathcal{A}_\square[\mathcal{T}]$. A set X *strictly self-assembles* if there is a TAS $\mathcal{T}$ for which every assembly $\alpha \in \mathcal{A}_\square[\mathcal{T}]$ satisfies $\text{dom } \alpha = X$. Note that if X strictly self-assembles, then X weakly self-assembles; let $B = T$.

Chapter 3. Random Number Selection in Self-Assembly

This chapter is joint work with Jack Lutz, Matt Patitz, Scott Summers, and Damien Woods and was originally published as [13].

3.1 Introduction

One prominent feature of self-assembly is its inherent nondeterminism, and the Tile Assembly Model represents this well. At any given time, there may be many locations where a tile might attach itself to the growing assembly, and, even at a single location, there may be more than one type of tile that can attach itself. When we are designing a tile assembly system that is supposed to result in a specified terminal assembly, regardless of the particular sequence in which tiles attach themselves, this nondeterminism is a conceptual difficulty that we overcome with tools like local determinism [38] and modularity [21]. In other situations, such as when we are using randomized algorithms to reduce the number of tile types required for a self-assembly [12, 19] or when we are simulating a system that is itself nondeterministic, the Tile Assembly Model's nondeterminism is a programming resource.

This chapter concerns the problem of using the nondeterminism inherent in self-assembly to implement a nondeterministic choice among an arbitrary number of options. That is, we consider the problem of designing a tile assembly system that, given a positive integer n, chooses an integer $r \in \{0, \ldots, n-1\}$.

As stated so far, this problem is not interesting. We simply count down from $n-1$ to 0, tossing a "coin" (implemented by having either one of two tiles attach at a suitable location) after each decrement to decide whether to stop or keep counting down. This solves the *logical* problem of nondeterministic choice, but, if n is large and the "coins" are fair and independent,

it is nearly certain that r will be much larger than 0. This is not satisfactory if we want to use the nondeterministic choice for a randomized algorithm or a useful simulation.

So our actual problem treats nondeterminism probabilistically. We assume that, at each time and location in self-assembly, all the tile types than *can* attach at that location are *equally likely* to do so, and that the "choices" made at different locations are independent. We then seek to design a tile assembly system that, given a positive integer n (represented in binary as a seed assembly), chooses an integer $r \in \{0, \ldots, n-1\}$ in such a way that the outcomes $r = 0, r = 1, \ldots, r = n-1$ are all equally likely, or nearly so. (We are *not* constructing pseudorandom generators in the sense of complexity theory or cryptography. Pseudorandom generators expand a short, truly random "seed" into a longer pseudorandom string. Our *random number selectors* use at least as much randomness as they produce.)

We present three solutions to this problem. Our first random number selector has 324 tile types and uses only $O(\log^2 n)$ space to select the number r, but the probabilities are only $\Theta(\text{uniform})$, in the sense that the probability that r is selected is between $\frac{1}{2n}$ and $\frac{2}{n}$.

Our second random number selector has 821 tile types and takes as input both n and a user-supplied precision parameter t. The probability that it selects r is then between $\frac{1}{n} - \frac{1}{t}$ and $\frac{1}{n} + \frac{1}{t}$. This is very nearly uniform if $t \gg n$, but this added uniformity comes at a price, namely that the selector uses $O(t^2)$ space.

Our third selector has 100 tile types and selects r with probability *exactly* $\frac{1}{n}$. With high probability it uses only as much space as our first selector, but there is *no* absolute bound on the space that it takes. (This selector is a self-assembly implementation of von Neumann's *rejection method* [20, 42].)

We expect each of these random number selectors to be useful in some self-assemblies. Taken together, our selectors suggest that there is a tradeoff between how uniform a random number selector is and how much space it takes. We conjecture that this tradeoff is real, and not merely an artifact of our constructions. Proving or disproving this conjecture is an open problem.

3.2 A Θ(uniform) Selector

The first selector that we implement chooses a number at random between $s \in \mathbb{N}$ ("start") and $e \in \mathbb{Z}^+$ ("end"), where $s < e$ and both are encoded in binary in the seed as shown in the bottom row of Figure 3.1.

The construction is shown in Figure 3.1. The tile set implements the random binary search algorithm SELECTOR. It is written such that each numbered line corresponds exactly to a row in Figure 3.1. The identifier ¢, wherever it appears, indicates the result of a fair coin flip, with result H or T. Each evaluation of ¢ is independent of the others.

```
SELECTOR({s, ..., e})
1  s' ← s + 1
2  while s' < e
3        do sum ← s + e
4           mid ← sum >> 1          ▷ divide sum by 2 by bit shifting
5           if ¢ = H
              then s ← mid + 1      ▷ choose right subinterval
              else e ← mid          ▷ choose left subinterval
6           s' ← s + 1
7  if ¢ = H then return s else return e
```

Figure 3.1 depicts a selector that randomly chooses an integer between s and e, such that, letting $n = |\{s, \ldots, e\}|$, we have $\frac{1}{2} \cdot \frac{1}{n} \leq \Pr(a) \leq 2 \cdot \frac{1}{n}$ for each $a \in \{s, \ldots, e\}$. The bottom row encodes s and e in binary. Note that the direction of growth can be inferred from the binding strengths. Each gray row is a comparison row whose result indicates whether the binary search is complete. The rows between each adjacent pair of gray rows implement the main loop of the random binary search. The interval $\{s, \ldots, e\}$ is subdivided into two subintervals, with the left subinterval always chosen to be one larger if $\{s, \ldots, e\}$ has an odd number of elements. One of the subintervals is randomly chosen to become the new interval, by replacing either s or e with a value near their midpoint. When $\{s, \ldots, e\}$ has either 1 or 2 elements, the loop

terminates, and one of s or e is randomly chosen.

Theorem 3.2.1. *For all $n \in \mathbb{N}$ and $r \in \{1, \ldots, n\}$,*

$$\frac{1}{2n} \leq \frac{1}{2^{\lfloor \log_2 n \rfloor + 1}} \leq \Pr[\textsc{Selector}(1, n) = r] \leq \frac{1}{2^{\lfloor \log_2 n \rfloor}} < \frac{2}{n}$$

Proof. Let $2^{i-1} < n \leq 2^i$, $i \in \mathbb{N}$. Immediately after k coin flips we are working within one of 2^k subintervals of $[1, n]$ that are non-overlapping and cover all of $[1, n]$. The leftmost subinterval, denoted $[1, n_k]$, has maximal size (number of elements) out of the 2^k sub-intervals, which can be shown by the following simple inductive argument. For $k = 1$, we have two sub-intervals and the leftmost is maximal since

$$\left\lfloor \frac{1+n}{2} \right\rfloor \geq n - \left\lfloor \frac{1+n}{2} \right\rfloor. \tag{3.2.1}$$

If we assume that the leftmost interval $[1, n_k]$ is maximal at flip k, then by letting $n = n_k$ in the inequality (3.2.1), the leftmost interval is maximal at flip $k + 1$.

So a maximal length sequence of coin flips, over all such valid sequences, is given by a sequence of 1's (we choose the left sub-interval at each coin flip). We next show that this maximal number of coin flips is no more than $\lfloor \log_2 n \rfloor + 1$. The leftmost interval $[1, n_k]$ has size

$$n_k = \left\lfloor \frac{1 + n_{k-1}}{2} \right\rfloor = \left\lceil \frac{n_{k-1}}{2} \right\rceil \tag{3.2.2}$$

$$= \begin{cases} \frac{n_{k-1}}{2} & n_{k-1} \text{ even} \\ \frac{n_{k-1}+1}{2} & n_{k-1} \text{ odd} \end{cases} \tag{3.2.3}$$

Assuming that n_k is odd for all k (to give an upper bound on the number of flips), we have $n_{\lfloor \log n \rfloor + 1} = 1$. This gives

$$\frac{1}{2^{\lfloor \log_2 n \rfloor + 1}} \leq \Pr[\textsc{Selector}(1, n) = r].$$

Similarly, using (3.2.1), it can be shown that at flip k, the rightmost subinterval is of minimal size over all 2^k possible subintervals. A lower bound of $\log_2 n$ on the number of flips is found when n_k is even for all k in (3.2.3), and this lower bound is reached when $n = 2^i$, $i \in \mathbb{N}$. Thus

$$\Pr[\textsc{Selector}(1, n) = r] \leq \frac{1}{2^{\lfloor \log_2 n \rfloor}}.$$

□

Unfortunately,the bounds of Theorem 3.2.1 are tight, in the sense that for certain n and $r \in \{0, \ldots, n-1\}$, $\Pr[r]$ can be nearly twice or half of $\frac{1}{n}$. The asymmetry of the interval sizes is what prevents the tile set from achieving perfect uniformity. Intuitively, if the left subinterval has length l and the right has length $l + 1$, then we should pick the right with probability $\frac{l+1}{2l+1}$, as opposed to $\frac{1}{2}$ as our algorithm does. It may seem at first glance that by randomly selecting which interval is larger, we could smooth out the probabilities and approach a uniform distribution in the limit as the interval size approaches infinity.

However, this does not work. Roughly speaking, selecting r from $\{0, \ldots, n-1\}$ using this method, we can express $\Pr[r] = \prod_{i=1}^{\approx \log n} p_i$, where each p_i represents the probability that r is in the subinterval picked during the i^{th} stage. If n is a power of 2, then each p_i will be $\frac{1}{2}$, and the selector (both that just described and the selector constructed earlier in this section) will select r with probability exactly $\frac{1}{n}$. If n is not a power of 2, then *at most* one of the p_i's will be unequal to $\frac{1}{2}$; this will occur precisely at the stage when the interval length is odd and r is the middle element, which can happen at most once in the course of the algorithm. In the case of $r = 1$ or $r = n - 2$, when this occurs, the interval will have length 3, so p_i will be equal to $\frac{1}{2}$, when in fact it ought to be $\frac{2}{3}$ to achieve uniformity. Therefore, no matter how large n is, $r = 1$ will be selected with probability a constant factor away from uniform.

In the next section we introduce a random selector that allows the user to *control* the desired closeness to uniformity as a parameter, trading off space for error in approximating uniformity.

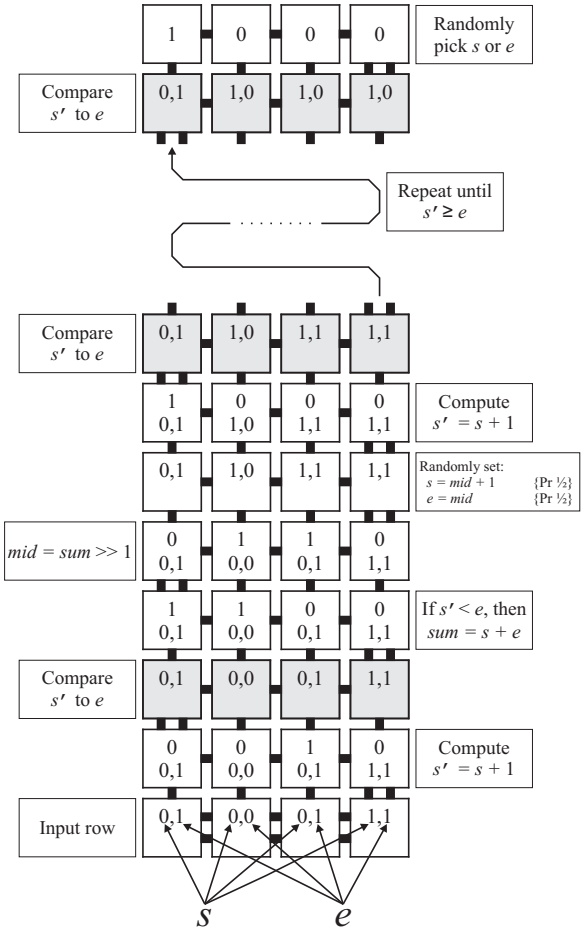


Figure 3.1: Θ(uniform) selector

3.3 A Controllable-Error Selector

In this section, we implement the self-assembly version of a random number generator whose deviation from uniform can be controlled by a user-specified parameter. More precisely, given an upper bound $n \in \mathbb{N}$, and a precision parameter $t \in \mathbb{N}$, the tile system generates a random number $r \in \{0, \ldots n-1\}$ with probability approaching $\frac{1}{n}$ as $t \to \infty$. We first outline the algorithm.

The following algorithm is one of the more intuitive methods of generating a random number from an arbitrary range, using flips of an unbiased coin, although the number is not generated with perfect uniformity. MSB stands for most significant bit, and LSB stands for least significant bit.

CONTROLLABLE-SELECTOR(n,t)

1 Uniformly at random choose $m \in \{0, \ldots, t-1\}$

2 **return** MOD (m, n)

MOD(m,n)

▷ compute m mod n in binary

1 Line up MSBs of m and n by shifting n to the left

$sm \leftarrow$ the integer represented by the $\lfloor \log n \rfloor + 1$ MSBs of m ("small m")

2 **while** n's LSB lies to the left of m's LSB

do if $sm \geq n$

3 **then** $sm \leftarrow sm - n$

concatenate the bit of m to the right of n's LSB to the end of sm

4 shift n one bit to the right relative to sm and m

5 **if** $sm \geq n$

6 **then** $sm \leftarrow sm - n$

7 **return** sm

As before, we have generally numbered the lines corresponding to rows in the tile assembly. However, in this case, lines 2 and 4 are implemented in the same row, since it is possible to shift

n to the right by one bit and simultaneously compare it to sm. This occurs on all comparisons except for the first check of the loop condition, immediately after n has been shifted *left* to line it up with the MSB of m. Also, line 3 represents the row that subtracts $sm - n$ if $sm \geq n$, or simply copies sm otherwise, although the copying does not appear as an explicit line in the pseudocode. Finally, line 1 is actually implemented as a series of rows as shown in Figure 3.3, but we express it as a single instruction in the pseudocode.

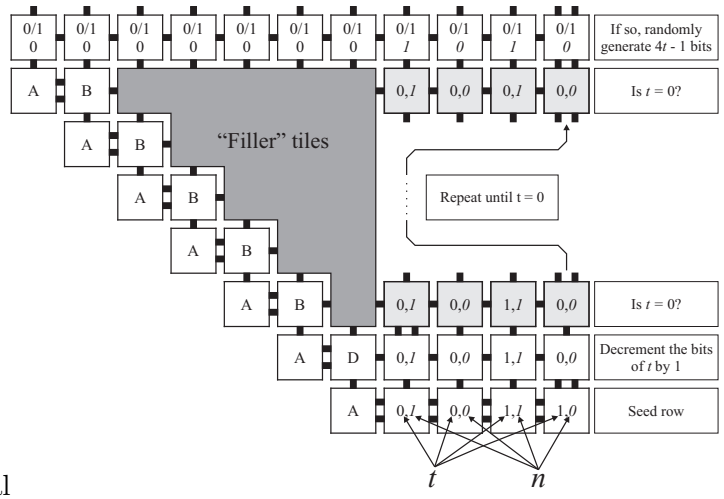
Our construction takes as inputs two positive integers, n and t', which we encode in a seed row. Choosing t' so that $t = 2^{4t'-1}$ means that the construction executes the procedure CONTROLLABLE-SELECTOR(n,t).[1] The construction first grows a number of rows upward, each having a width one greater than the previous, until a row having a width of exactly $4t' - 1$ is formed. Finally, the top most row assembles in which, at each location, one of exactly two tile types is randomly selected to bind. This effectively generates a random number m in the set $\left\{0, \ldots 2^{4t'-1} - 1\right\}$ with a uniform distribution. Then, the subsequent rows to attach implement step 2 in CONTROLLABLE-SELECTOR. The resulting modulus is encoded in the top most row of the construction and is the number r.

We now present a more detailed discussion of this construction. The tile assembly system can be thought of as assembling in three logical stages.

First Stage: In the initial stage, our goal is to take as input n and t', and (uniformly) produce a random number, say m, in the range $\left\{0, \ldots 2^{4t'-1} - 1\right\}$. We will then feed the numbers m and n to the second stage of the construction. Our input is encoded in a seed row whose length k is the greater of $\lfloor \log n \rfloor + 1$ and $\lfloor \log t' \rfloor + 1$. The seed row is the bottom most row in Figure 3.2.

1. On top of the seed row, we use a (zig-zagging) Rothemund-Winfree binary subtractor [32] that counts down from t' while "blindly" passing the value of n up through the subsequent rows. This gives us a k by $2t' + 1$ rectangle.

[1] If $t \neq 2^{4t'-1}$ for some $t' \in \mathbb{N}$, choose t' so that $t \leq 2^{4t'-1}$. This achieves the bound on the probability that we derive for t, while using asymptotically the same amount of space.



normal

Figure 3.2: The first stage of the construction.

2. We use filler tiles (labeled 'A', 'B' and "Filler" tiles in Figure 3.2) to self-assemble a right-triangle to the left of the rectangle that was built in the previous step.

3. (line 1 of the pseudocode for CONTROLLABLE-SELECTOR) The top most row of the construction is $4t' - 1$ tiles long and self-assembles from right to left such that, for each of the $4t' - 1$ locations in the row, there are two possible tile types that can attach - one tile representing a 0 and the other a 1. The choice of each tile that attaches in this row can be made independently of its predecessor (i.e., the tile immediately to its right). This results in the generation of a random number m in the range $\left\{0, \ldots 2^{4t'-1} - 1\right\}$ with uniform distribution.

The top of the last row of the initial stage of the construction encodes (as output) the number n (the "lower" bits in each tile), along with the number m (the randomly chosen "upper" bits in each tile). Note that we allow leading zeros in both m and n.

Second Stage: (Line 1 of the pseudocode for MOD) The second stage of the construction takes as input m and n (output from the previous stage), and shifts (all of the bits of) n to the left so that the most significant 1 of n lines up with the MSB of m.

1. Every other row (starting with the first row) in this stage of the construction sends a "request-a-shift" signal from left-to-right only if the most significant 1 of n does not line up with the MSB of m. For example, the second (from the bottom most) row in Figure 3.3 self-assembles left-to-right and carries the first request-a-shift signal.

2. On top of each row that carries a request-a-shift signal, a row self-assembles right-to-left in which each bit of n is shifted once to the left. The third (from the bottom most) row in Figure 3.3 is the first row that carries out a shift operation. For shift rows, as self-assembly proceeds right-to-left, the position of the most significant 1 of n is compared with the MSB of m, and if they line up, then a subsequent request-a-shift signal is not sent. It is at this point that the third, and final stage of the construction is carried out.

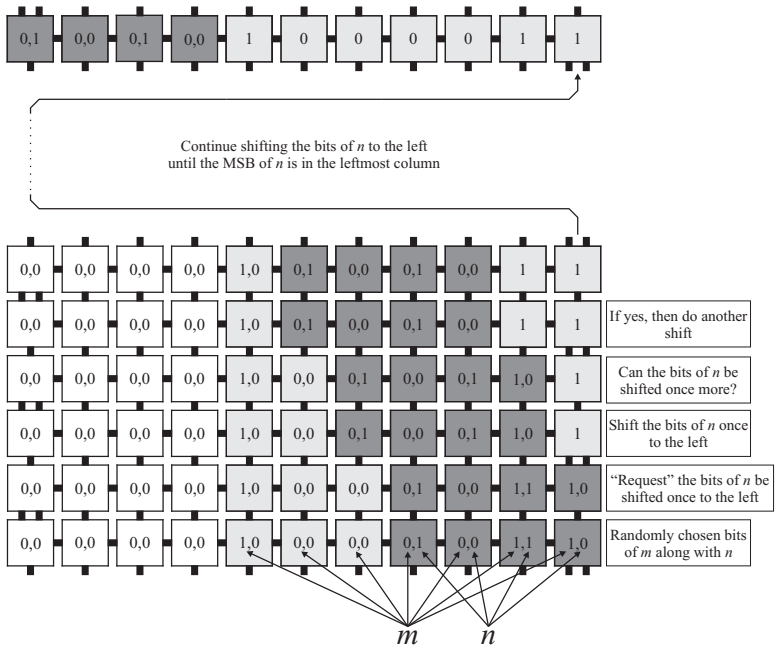


Figure 3.3: The second stage of the construction.

Third Stage: The third stage of the construction performs the task of calculating $m \mod n$ (step 2 in Controllable-Selector) and then (as a final step) self-assembles a row

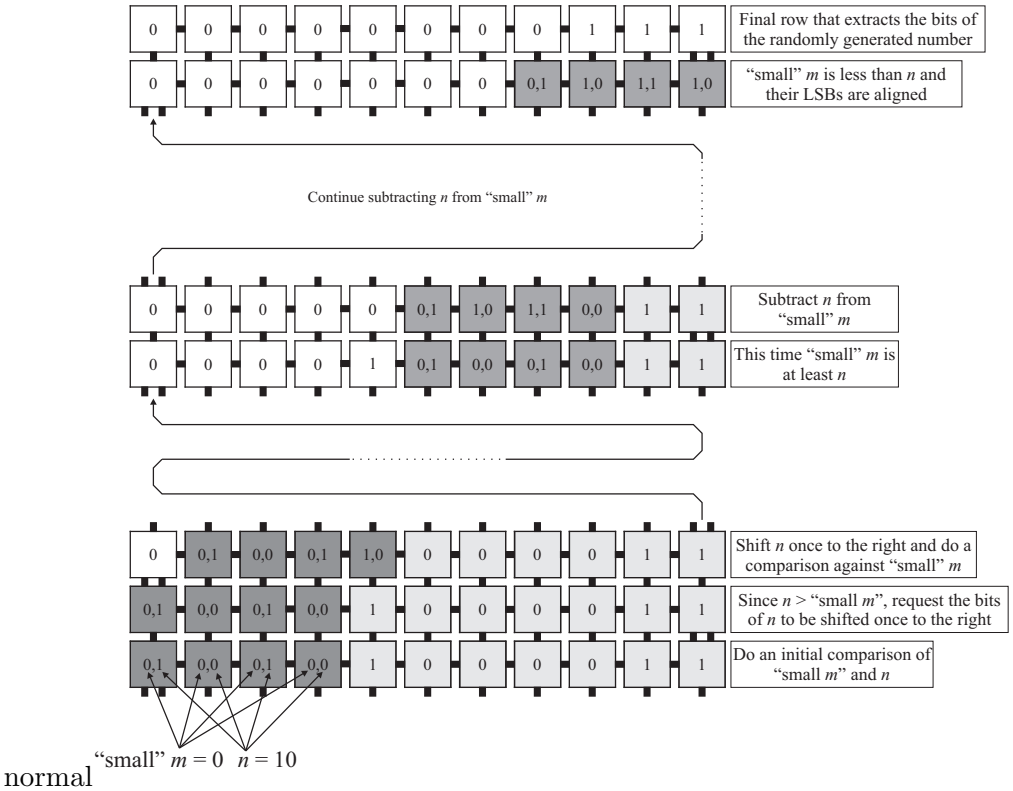


Figure 3.4: The third stage of the construction. The "direction" of self-assembly for each row is either right-to-left or left-to-right (as noted in parentheses).

which represents only that value as the final output of the construction. The third stage begins with a row of tiles which encode the values of m and n such that the most significant 1 of n lines up with the MSB of m. Throughout this discussion, "small m" will refer to the bits of m that begin with the leftmost and continue right to the location of the rightmost bit of n in that row (these positions are represented by the dark colored tiles in Figure 3.4). The third stage proceeds as follows.

1. (line 2 of the pseudocode for MOD) Self-assemble a row from left-to-right which performs an initial comparison of the values of "small m" and n.
2. (line 3 of the pseudocode for MOD) Self-assemble the next row from right to left which performs one of the following functions based on the comparison from the previous step.
   (a) If $n >$ "small m," send a signal requesting that (all of the bits of) n be shifted one position to the right.
   (b) Else, subtract the value of n from "small m" request to shift n. Note that this will result in a new value of "small m."
3. (line 4 of the pseudocode for MOD) Now the construction self-assembles a row from left-to-right which shifts the value of n one position to the right (shifting 0 bits in from the left) and keeping the current value of "small m" in the same position. Note that this row also performs a comparison against the newly shifted value of n and the current value of "small m." The latter will now extend one bit further to the right than it previously did.
4. (lines 2 and 4 of the pseudocode for MOD) The construction continues the self-assembly of rows which perform the previous two steps in a loop until the value of n has shifted far enough to the right so that its LSB is aligned with that of m.
5. (line 6 of the pseudocode for MOD) Only if the last comparison resulted in $n <$ "small m," the construction self-assembles the necessary rows to do one more subtraction of n from "small m," and a final comparison (as done above).

6. (line 7 of the pseudocode for MOD) The remaining value of "small m" now represents the result of $m \mod n$. The final step in the third stage of the construction self-assembles a row from right-to-left which represents only that remainder, and thus the final output, $r \in \{0, \ldots, n-1\}$.

Theorem 3.3.1. *For all $n \in \mathbb{N}$, and $r \in \{0, \ldots, n-1\}$,*

$$\lim_{t \to \infty} \Pr\left[\textsc{Controllable-Selector}(n, t) = r\right] = \frac{1}{n}$$

Proof. Fix $t \in \mathbb{N}$. Note that, by a trivial counting argument, there are either $\left\lfloor \frac{t}{n} \right\rfloor$ or $\left\lceil \frac{t}{n} \right\rceil$ ways to choose a number congruent to r (modulo n) from the set $\{0, \ldots, t-1\}$, whence

$$\begin{aligned} \Pr\left[\textsc{Controllable-Selector}(n, t) = r\right] &\leq \frac{\left\lceil \frac{t}{n} \right\rceil}{t} \\ &\leq \frac{\frac{t}{n} + 1}{t} \\ &= \frac{1}{n} + \frac{1}{t}. \end{aligned}$$

Similarly, we have

$$\Pr\left[\textsc{Controllable-Selector}(n, t) = r\right] \geq \frac{1}{n} - \frac{1}{t}.$$

It follows, by the Squeeze theorem, that

$$\lim_{t \to \infty} \Pr\left[\textsc{Controllable-Selector}(n, t) = r\right] = \frac{1}{n}.$$

□

3.4 An Exactly Uniform Selector

Even though the selector of Section 3.3 asymptotically approaches a uniform distribution as the precision parameter t grows to infinity, for certain n it deviates slightly from uniform.

In this section we outline a construction of a random number selector with the property

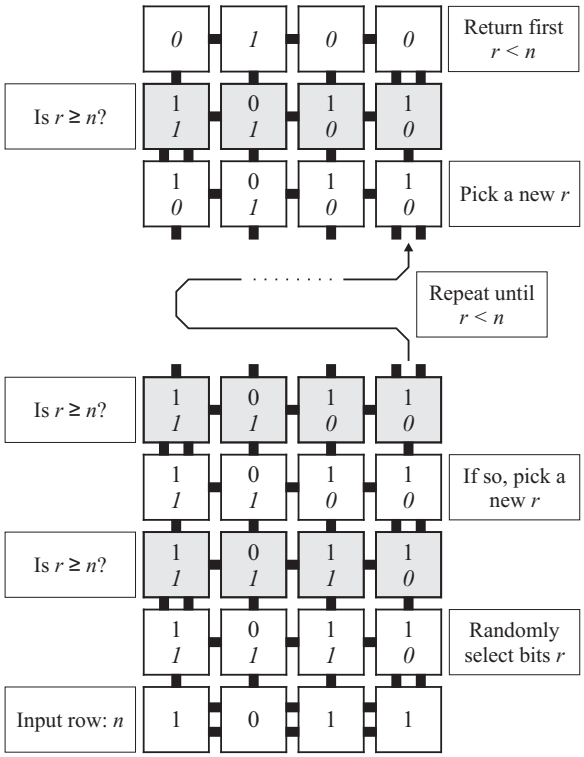


Figure 3.5: The uniform selector

that it selects a random number in the range $\{0, \ldots, n-1\}$ with probability exactly $\frac{1}{n}$. The catch is that while the selector has a very low expected number of tiles that need to be used, with some small probability, an unbounded number of tiles could be required, making this selector unsuitable for applications in which space constraints must absolutely be enforced.

The construction is shown in Figure 3.5. The tile set implements the following algorithm. The random element r is generated by selecting t bits uniformly and independently at random, where t is the length of the binary expansion of n.

UNIFORM-SELECTOR(n)

1 $r \leftarrow$ random element of $\{0, 1, \ldots, 2^{\lfloor \log n \rfloor + 1} - 1\}$

2 **while** $r \geq n$

3 **do** $r \leftarrow$ random element of $\{0, 1, \ldots, 2^{\lfloor \log n \rfloor + 1} - 1\}$

4 **return** r

The selector chooses r between 0 and the next power of 2 above n and outputs r if $r < n$. The selector will use a small number of rows with high probability, but may potentially use an unbounded number of rows.

Since the element of $\left\{0, 1, \ldots, 2^{\lfloor \log n \rfloor + 1} - 1\right\}$ is selected with uniform probability, then conditioned on that element being less than n, the probability of each element $r \in \{0, \ldots, n-1\}$ is equal, i.e., $\frac{1}{n}$. Furthermore, since the next power of two greater than n is at most $2n$, the probability that $r \geq n$ is at most $\frac{1}{2}$, whence the expected number of iterations i is a geometric random variable with expected value at most 2, subject to the tail bound, for all $k \in \mathbb{N}, \Pr[i > k] \leq 2^{-k}$.

Since two rows are used per iteration, so allowing, for instance, 100 rows (plus the two for the seed and final row) ensures that sufficient room will exist to generate r with probability at least $1 - 2^{-50}$.

3.5 Conclusion

We have introduced random number selectors in the Tile Assembly Model, powered by the randomness inherent in nondeterministic tile sets. They allow for a more algorithmic control

of randomness than by hard-coding probabilities by fixed tile concentrations.

The perfectly uniform selector is not entirely unlike the procedure used to generate random integers from random bits used in many standard libraries; for instance, the method `Random.nextInt` in the Java standard library. Why then have we bothered to describe the first two selectors, which are provably different from uniform? In a programming language, a "ZPP" algorithm that almost certainly takes a small amount of time, but may rarely use more time, is not a problem except perhaps in critical real-time systems. However, in a tile assembly system, consisting of possibly billions of tiles, the presence of even a single region that uses too much space will destroy the correctness of the entire assembly. In such situations, it may be preferable to guarantee that space bounds are adhered to at the cost of a slight deviation from uniformity.

Chapter 4. Randomized Self-Assembly for Exact Shapes

This chapter was originally published as [12].

4.1 Introduction

Rothemund and Winfree [32] investigated the minimum number of tile types needed to uniquely assemble an $n \times n$ square. Utilizing the theory of Kolmogorov complexity, they show that for any algorithmically random n, $\Omega\left(\frac{\log n}{\log \log n}\right)$ tile types are required to uniquely assemble an $n \times n$ square, and Adleman, Cheng, Goel, and Huang [1] exhibit a construction showing that this lower bound is asymptotically tight.

Real-life implementations of the aTAM involve (at the present time) creating tile types out of DNA double-crossover molecules [33], copies of which can be created at an exponential rate using the polymerase chain reaction (PCR) [34]. PCR technology has advanced to the point where it is automated by machines, meaning that *copies* of tiles are easy to supply, whereas the number of distinct tile *types* is a precious resource, costing much more lab time to create. Therefore, effort has been put towards developing methods of "programming" tile sets through methods other than hard-coding the desired behavior into the tile types. Such methods include *temperature programming* [18, 41], which involves changing the ambient temperature through the assembly process in order to alter which bonds are possible to break or create, and *staged assembly* [11], which involves preparing different assemblies in different test tubes, which are then mixed after reaching a terminal state. Each of these models allows a *single* tile set to be reused for assembling different structures by programming it with different environmental conditions affecting the behavior of the tiles.

The model used in this chapter is known as *tile concentration programming*. If the tile

assembly system is nondeterministic – if intermediate assemblies exist in which more than one tile type is capable of binding to the same position – and if the solution is well-mixed, then the relative concentrations of these tile types determine the probability that each tile type will be the one to bind.

Tile concentrations affect the expected time before an assembly is completed (such a model is considered in [1] and [5], for instance), but we ignore such running time considerations in the present chapter. We instead focus on using the biased randomness of tile concentrations to guide a probabilistic shape-building algorithm, subject a certain kind of "geometric *space* bound"; namely, that the algorithm must be executed within the confines of the shape being assembled. This restriction follows from the monotone nature of the aTAM: once a tile attaches to an assembly, it never detaches. Chandran, Gopalkrishnan, and Reif [8] show that a one-dimensional line of expected length n can be assembled using $\Theta(\log n)$ tile types, subject to the restriction that all tile concentrations are equal. Furthermore, they show that this bound is tight for *all* n. Becker, Rapaport, and Rémila [5] show that there is a *single* tile assembly system $\mathcal{T}$ such that, for all $n \in \mathbb{Z}^+$, setting the tile concentrations appropriately causes $\mathcal{T}$ to assemble an n' x n' square, such that n' has expected value n. However, n' will have a large deviation from n with non-negligible probability. Kao and Schweller [19] improve this result by constructing, for each $\delta, \epsilon > 0$, a tile assembly system $\mathcal{T}$ such that setting the tile concentrations appropriately causes $\mathcal{T}$ to assemble an n' x n' square, where $(1-\epsilon)n \leq n' \leq (1+\epsilon)n$ with probability at least $1-\delta$, for sufficiently large $n \in \mathbb{Z}^+$.

Kao and Schweller asked whether a constant-sized tile assembly system could be constructed that, through tile concentration programming, would assemble a square of dimensions *exactly* $n \times n$, with high probability. We answer this question affirmatively, showing that, for each $\delta > 0$, there is a tile assembly system $\mathcal{T}$ such that, for sufficiently large $n \in \mathbb{Z}^+$, there is an assignment of tile concentrations to $\mathcal{T}$ such that $\mathcal{T}$ assembles an $n \times n$ square with probability at least $1-\delta$.

Kao and Schweller also asked whether arbitrary finite connected shapes, possibly scaled by factor $c \in \mathbb{N}$ (depending on the shape) by replacing each point in the shape with a $c \times c$ block

of points, could be assembled from a constant tile set through concentration programming. Our construction answering the first question computes the binary expansion of n with high probability in a self-assembled rectangle of height $O(\log n)$ and width $O(n^{2/3})$. By assembling this structure within the "seed block" of the construction of [38], our construction can easily be combined with that of [38] to answer this question affirmatively as well, by replacing the number n with a program that outputs a list of points in the shape, and using this as the "seed block" of the construction of [38].

This chapter is organized as follows. Section 4.2 defines tile concentration programming. Section 4.3 provides the construction and proof of correctness. Section 4.4 concludes the chapter, discusses practical limitations of the construction and potential improvements, and suggests non-square structures that can be assembled with the same technique.

4.2 Tile Concentration Programming

Let $\mathcal{T} = (T, \sigma, \tau)$ be a TAS. A *tile concentration assignment* on $\mathcal{T}$ is a function $\rho : T \to [0, \infty)$.[1] If $\rho(t)$ is not specified explicitly for some $t \in T$, then $\rho(t) = 1$. If $\alpha : \mathbb{Z}^2 \dashrightarrow T$ is a τ-stable assembly such that $t_1, \ldots, t_j \in T$ are the tiles capable of binding to the same position $\vec{m}$ of α,[2] then for $1 \leq i \leq j$, t_i binds at position $\vec{m}$ with probability $\frac{\rho(t_i)}{\rho(t_1)+\ldots+\rho(t_j)}$.[3] ρ induces a probability measure on $\mathcal{A}_\square[\mathcal{T}]$ in the obvious way.[4] For $p \in [0, 1]$ and $X \subseteq \mathbb{Z}^2$, we say X *strictly self-assembles in* $\mathcal{T}(\rho)$ *with probability at least* p if $\Pr_{\alpha \in \mathcal{A}_\square[\mathcal{T}]}[\text{dom } \alpha = X] \geq p$. That is, $\mathcal{T}$ self-assembles into a shape equal to X with probability at least p. Note that different two assemblies may have the same shape though they might assign different tile types to the

[1] Note in particular that we do not require ρ to be a probability measure on T. ρ induces a probability measure as described later on subsets of tiles in T that contend nondeterministically to bind, but there may be more than one such subset, and the relative concentration of a tile from one subset to that of another is irrelevant, since they do not compete.

[2] More precisely, if $\alpha + (\vec{m} \mapsto t_i)$ is τ-stable for some $\vec{m} \notin \text{dom } \alpha$ and all $i \in \{1, \ldots, j\}$, but $\alpha + (\vec{m} \mapsto t)$ is not τ-stable for any $t \in T - \{t_1, \ldots, t_j\}$.

[3] This quantity is the *conditional* probability that t_i attaches to position $\vec{m}$, *given* that one of $t_1, \ldots, t_j$ will bind to position $\vec{m}$ at the current stage of self-assembly. We do not use probabilities to model the choice of which position $\vec{m}$ receives a tile; any fair assembly sequence (see [22] for a definition) will suffice.

[4] Formally, let $\alpha \in \mathcal{A}_\square[\mathcal{T}]$ be a producible terminal assembly. Let $A(\alpha)$ be the set of all assembly sequences $\vec{\alpha} = (\alpha_i \mid 1 \leq i \leq k)$ such that $\text{res}(\vec{\alpha}) = \alpha$, with $p_{\vec{\alpha},i}$ denoting the probability of attachment of the tile added to α_{i-1} to produce α_i (noting that $p_{\vec{\alpha},i} = 1$ if the i^{th} tile attached without contention). Then $\Pr[\alpha] = \sum_{\vec{\alpha} \in A(\alpha)} \prod_{i=2}^{k} p_{\vec{\alpha},i}$.

same position.

4.3 Construction of the Tile Set

This section is devoted to proving the following theorem, which is the main result of this chapter.

For all $\delta > 0$ and $n \in \mathbb{Z}^+$, define $r_\delta = \left\lceil \frac{\log \frac{\delta}{8}}{\log 0.9421} \right\rceil$, $c_\delta = 2 + \left\lceil \log \frac{\log \frac{\delta}{8}}{\log 0.717} \right\rceil$, and $k_n = \left\lceil \frac{\lfloor \log n \rfloor + 1}{3} \right\rceil$, and define $b(n, \delta) = \max\left\{r_\delta, 2^{2k_n + c_\delta}\right\} + c_\delta + 3k_n$.

Theorem 4.3.1. *For all $\delta > 0$, there is a tile assembly system $\mathcal{T}_\delta = (T, \sigma, 2)$ such that, for all integers $n \geq b(n, \delta)$, there is a tile concentration assignment $\rho_n : T \to [0, \infty)$ such that a translation of the set $\{\ (x, y) \mid x, y \in \{1, \ldots, n\}\ \}$ strictly self-assembles in $\mathcal{T}_\delta(\rho_n)$ with probability at least $1 - \delta$.*

Note that for any fixed $\delta > 0$, $b(n, \delta) = O(n^{2/3})$ (the constant in the $O()$ depending on δ), whence $n \geq b(n, \delta)$ for all sufficiently large n.

4.3.1 Intuitive Idea of the Construction

Kao and Schweller introduced a basic primitive in [19] (refining a lower-precision technique described in [5]), called a *sampling line*. The sampling line allows tile concentrations to encode a natural number whose binary representation can be probably approximately reproduced. Kao and Schweller utilize the sampling line to encode $n \in \mathbb{N}$ by an approximation $n' \in \mathbb{N}$ such that $(1 - \epsilon)n \leq n' \leq (1 + \epsilon)n$ with probability at least $1 - \delta$.

The idea of our construction is as follows. We will "approximate" only numbers m small enough that the sampling line approximation has sufficient space to be an *exact* computation of m with high probability. The construction of Kao and Schweller can be thought of as estimating n by, in a sense, probabilistically counting to n using independent Bernoulli trials with appropriately fixed success probability; i.e., the probabilities are used to estimate an approximate *unary* encoding of n, which is converted to binary by a counter. Representing n in unary, of course, takes space n, and recovering it probabilistically from tiles subject to

randomization requires using much more than space n to overcome the error introduced by randomization. Kao and Schweller use an ingenious technique to spread this estimation out into the center of the $n \times n$ square being built, affording $O(n^2)$ space to approximate n closely. However, that construction lacks the space to compute n exactly, which requires much more than n^2 Bernoulli trials – applying the standard Chernoff bound to the Kao-Schweller sampling line achieves an upper bound of $O(n^5)$ trials – to achieve a sufficiently small estimation error. Hence, attempting to use a sampling line directly to compute n would result in a line containing many more tiles than the n^2 tiles that compose an $n \times n$ square, and no amount of twisting the line will cause it to fit inside the boundaries of the square.

We split n's binary expansion $b(n) = b_1 b_2 \dots b_{\lfloor \log n \rfloor + 1} \in \{0,1\}^*$ into three subsequences $b_1 b_4 b_7 \dots$, $b_2 b_5 b_8 \dots$, and $b_3 b_6 b_9 \dots$, each of length about $\frac{1}{3} \log n$, and interpret these binary strings as natural numbers $m_1, m_2, m_3 \leq n^{1/3}$ to be estimated. The problem of estimating n is reduced to that of estimating these three numbers. At the same time, we introduce a new sampling line technique that can exactly estimate a number m with high probability using only $O(m^2)$ trials.[5] Since $m_1, m_2, m_3 \leq n^{1/3}$, estimating m_1, m_2, and m_3 will require $O(n^{2/3})$ trials, which fits within the width of an $n \times n$ square for sufficiently large n.

Intuitively, the reason that estimating m_1, m_2, and m_3 creates an improvement over estimating n directly is that the space needed for the unary encodings of numbers whose *binary* length is one-third that of n's does not scale linearly with that length; the *unary* encoding of these numbers scales with $n^{1/3}$, not $n/3$, whence a quadratic increase in the space needed for probabilistic recovery remains sufficiently small ($O(n^{2/3})$) that three such decodings easily fit into space n.

[5] As opposed to the $O(m^5)$ trials that would be required by the Kao-Schweller sampling line. It is possible to use Kao and Schweller's original sampling line to estimate seven numbers – $\lfloor \log n \rfloor + 1$ (the length of the binary expansion of n), and the six numbers m_1 - m_6 encoded by length-$\left\lceil \frac{\lfloor \log n \rfloor + 1}{6} \right\rceil$ substrings of n's binary expansion, each small enough that $m_i^5 = o(n)$ – and to use these numbers to reconstruct n and from that, build an $n \times n$ square. A straightforward and tedious analysis of the constants involved reveals that such a technique can be used to construct $n \times n$ squares for $n \geq 10^{18}$. We achieve much more feasible bounds on n ($\approx 10^7$ for $\delta = 0.01$) using the techniques introduced in this chapter, and indeed, better bounds than those required by Kao and Schweller to approximate n, whose construction achieves, for instance, a $(0.01, 0.01)$-approximation only for $n \geq 10^{13}$, according to their analysis.

4.3.2 Probabilistic Decoding of a Natural Number using a Sampling Line

In this section, we describe how to exactly compute a positive integer m probabilistically from tile concentrations that are appropriately programmed to represent m. In our final construction, the sampling line will estimate not one but three integers m_1, m_2, and m_3, as described in Section 4.3.1, by embedding additional bits into the tiles. However, for the sake of clarity, in this section, we describe how to estimate a single positive integer m, and then describe in Section 4.3.2.2 how to modify the construction and set the probabilities to allow three numbers to be estimated simultaneously on a single sampling line.

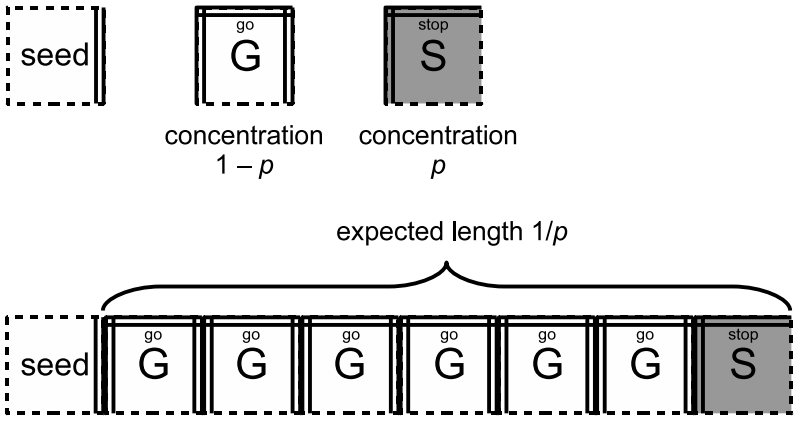


Figure 4.1: The portion of the basic Kao-Schweller sampling line that controls its length. Two tiles compete nondeterministically to bind to the right of the line, one of which stops the growth, while the other continues, giving the length of the line a geometric distribution.

The basic length-controlling portion of the Kao-Schweller sampling line is shown in Figure 4.1.[6] A horizontal row of tiles forms to the right of the seed. Two tiles, G ("go") and S ("stop") nondeterministically connect to the right end of the line; G continues the growth, while S stops the growth. If S has concentration $p \in [0, 1]$ and G has concentration $1 - p$, then the length L of the line is a geometric random variable with expected value $1/p$. By setting p appropriately, $\mathrm{E}[L]$ can be controlled, but not precisely, since a geometric random variable may have a deviation from the expected value that is too large for our purposes.

[6] Our description of the Kao-Schweller sampling line is incomplete, as discussed in the next paragraph.

Kao and Schweller allow a third tile type to bind within the sampling line, which does the actual sampling for computing a natural number, but our construction splits this sampling into a separate set of tiles that forms above the line. The sampling portion is discussed in Section 4.3.2.2. For the present time, we restrict our discussion to controlling the length of the line.

4.3.2.1 More Precisely Controlling the Sampling Line Length

Our goal is to control L, the length of the sampling line, such that, by setting tile concentrations appropriately, we may ensure that L lies between 2^{a-1} and 2^a with high probability, for an $a \in \mathbb{Z}^+$ of our choosing (which will be influenced by the number n we are estimating). That is, we may ensure that the number of bits required to represent L is computed precisely, even if the exact value of L varies widely within the interval $[2^{a-1}, 2^a)$. We then attach a counter – a group of tiles that measures the length of the line by counting in binary – to the north of the line that measures L until the final stopping tile. The stop signal is not intended to stop the counter immediately, but rather to signal that the counter should continue until it reaches the next power of 2 – i.e., the next time a new most significant bit is required – and then stop. Hence, we may choose an arbitrary power of 2 and set tile concentrations to ensure that the counter counts to that value and then stops.

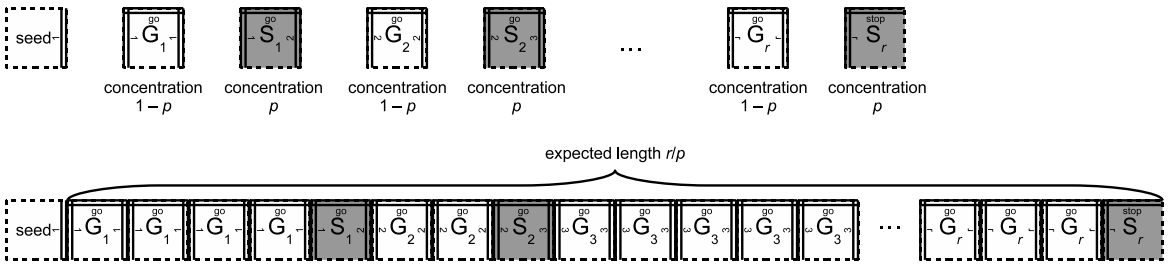


Figure 4.2: The portion of the sampling line of our construction that controls its length. r stages each have expected length $1/p$, making the expected total length r/p, but more tightly concentrated about that expected length than in the case of one stage.

To increase the precision with which we control L, we use not one but many stages of "go" and "stop" tiles, $G_1, S_1, G_2, S_2, \ldots, G_r, S_r$. The construction is shown in Figure 4.2. G_i and S_i

each compete to bind to the right of S_{i-1} and G_i. S_i signals a transition to the next stage $i+1$, with S_r stopping the growth of the line after r stages. Therefore, the sequence of tiles to the right of the seed is a string described by the regular expression $G_1^* S_1 G_2^* S_2 \ldots G_r^* S_r$. Each S_i has concentration p, and the remaining G_i tiles each have concentration $1-p$. The length L of the line is a *negative binomial* random variable[7] with parameters r, p (see [26]) with expected value r/p by linearity of expectation; i.e., its length is the number of Bernoulli trials required before exactly r successes, provided each Bernoulli trial has success probability p.

Let $N, R \in \mathbb{N}$ and $p \in [0,1]$. A binomial random variable $\mathcal{B}(N,p)$ (the number of successes after N Bernoulli trials, each having success probability p) is related to a negative binomial random variable $\mathcal{N}(R,p)$ (the number of trials before exactly R successes) by the relationships

$$\Pr[\mathcal{N}(R,p) < N] = \Pr[\mathcal{B}(N,p) > R] \text{ and} \tag{4.3.1}$$

$$\Pr[\mathcal{N}(R,p) > N] = \Pr[\mathcal{B}(N,p) < R]. \tag{4.3.2}$$

Thus, Chernoff bounds that provide tail bounds for binomial distributions can be applied to negative binomial distributions via (4.3.1) and (4.3.2).

To cause L to fall in the interval $[2^{a-1}, 2^a)$, we must set its expected length $\overline{L}$ (by setting $p = r/\overline{L}$) to be such that the r^{th} success occurs when the line has length in the interval $[2^{a-1}, 2^a)$. Note that pN is the expected number of successes in the first N tiles of the line; i.e., it is the expected number of successes in exactly N Bernoulli trials.

We define ϵ and ϵ' so that $\overline{L} = (1+\epsilon)2^{a-1} = (1-\epsilon')2^a$ and the two error probabilities derived below are approximately equal; $\epsilon \approx 0.442695$ and $\epsilon' \approx 0.2786525$ suffice. The event that $L < 2^{a-1}$ is equivalent to the event that 2^{a-1} Bernoulli trials are conducted (with expected number of successes $p2^{a-1}$) with at least r successes. By (4.3.1) and the Chernoff bound [26,

[7] The term *negative* is misleading; a negative binomial random variable is better described (informally) as the *inverse* of a binomial random variable, if one thinks of a binomial random variable as being like a function that maps a number of Bernoulli trials to a number of successes. A negative binomial random variable maps a number of successes to the number of trials necessary to achieve that number of successes.

Theorem 4.4, part 1],

$$\begin{aligned}\Pr\left[L < 2^{a-1}\right] &= \Pr\left[r > (1+\epsilon)p2^{a-1}\right] &\leq \left(\tfrac{e^{\epsilon}}{(1+\epsilon)^{1+\epsilon}}\right)^{p2^{a-1}} &= \left(\tfrac{e^{\epsilon}}{(1+\epsilon)^{1+\epsilon}}\right)^{r2^{a-1}/\overline{L}} \\ &= \left(\tfrac{e^{\epsilon}}{(1+\epsilon)^{1+\epsilon}}\right)^{r2^{a-1}/\left((1+\epsilon)2^{a-1}\right)} &= \left(\tfrac{e^{\epsilon}}{(1+\epsilon)^{1+\epsilon}}\right)^{r/(1+\epsilon)} &< 0.9421^{r}.\end{aligned}$$

The event that $L \geq 2^a$ is equivalent to the event that 2^a Bernoulli trials are conducted (with expected number of successes $p2^a$) with fewer than r successes. To bound the probability that L is too large, we use (4.3.2) and the Chernoff bound for deviations below the mean [26, Theorem 4.5, part 1],

$$\begin{aligned}\Pr\left[L \geq 2^{a}\right] &= \Pr\left[r \leq (1-\epsilon')p2^{a}\right] &\leq \left(\tfrac{e^{-\epsilon'}}{(1-\epsilon')^{1-\epsilon'}}\right)^{p2^{a}} &= \left(\tfrac{e^{-\epsilon'}}{(1-\epsilon')^{1-\epsilon'}}\right)^{r2^{a}/\overline{L}} \\ &= \left(\tfrac{e^{-\epsilon'}}{(1-\epsilon')^{1-\epsilon'}}\right)^{r2^{a}/\left((1+\epsilon)2^{a-1}\right)} &= \left(\tfrac{e^{-\epsilon'}}{(1-\epsilon')^{1-\epsilon'}}\right)^{2r/(1+\epsilon)} &< 0.9421^{r}.\end{aligned}$$

By the union bound,

$$\Pr[L \notin [2^{a-1}, 2^{a})] < 2 \cdot 0.9421^{r} \tag{4.3.3}$$

Therefore, by setting r sufficiently large, we can exponentially decrease the probability that L falls outside the range $[2^{a-1}, 2^{a})$, independently of a. For example, letting $r = 113$ leads to $\Pr\left[L \notin [2^{a-1}, 2^{a})\right] < 0.0025$. Since r is a constant depending only on δ, it can be encoded into the tile types as shown in Figure 4.2.

4.3.2.2 Computing a Number Exactly using a Sampling Line

As stated previously, our goal is that, with a sampling line of length $O(m^2)$, we can exactly compute a number m. The idea is shown in Figure 4.3, and is inspired by the sampling line of Kao and Schweller [19] but can estimate a number more precisely using a given length, as well as having a length that is itself controlled more precisely by the technique of Section 4.3.2.1. The length-controlling portion of the sampling line of length L will control a counter placed above the sampling line, which counts to the next power of 2 greater than L, 2^a. This counter will eventually end up with a total bits before stopping. Let k be the maximum number of

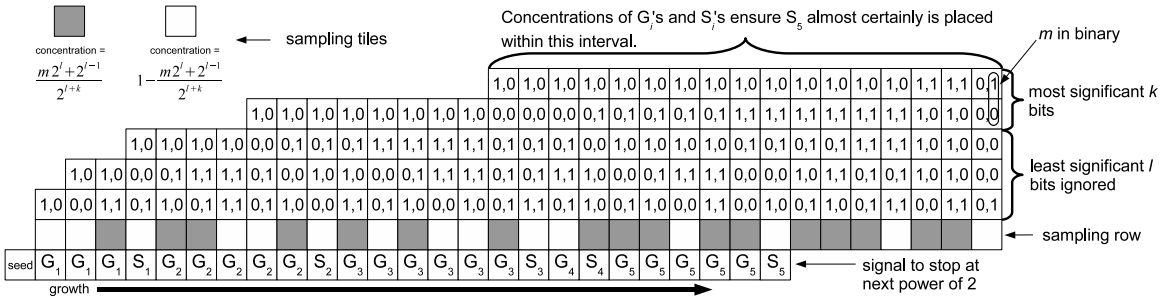


Figure 4.3: Computing the natural number $m = 2$ from tile concentrations using a sampling line. For brevity, glue strengths and labels are not shown. Each column increments the primary counter, represented by the bits on the left of each tile, and each gray tile increments the sampling counter, represented by the bits on the right of each tile. The number of bits at the end is $l + k$, where c is a constant coded into the tile set, and k depends on m, and $l = k + c$. The most significant k bits of the sampling counter encode m. In this example, $k = 2$ and $c = 1$.

bits needed to represent m (k will be about $\frac{1}{3}\log n$ in our application), and let $l = a - k$. We form a row above the row described in Section 4.3.2.1, which does the sampling. To implement the Bernoulli trials that estimate m, one of two tiles A (the gray tile in Figure 4.3) or B (the white tile in Figure 4.3) nondeterministically binds to every position of this row. Set the concentration of A to be $\frac{m2^l+2^{l-1}}{2^a}$ and the concentration of B to be $1 - \frac{m2^l+2^{l-1}}{2^a}$. We embed a second counter – the sampling counter – within the primary counter. Whenever A appears, the sampling counter increments, and when B appears it does not change. Let M be the random variable representing the final value of the sampling counter. Then M is a binomial random variable with $\mathrm{E}[M] = m2^l + 2^{l-1}$.

We will choose k and l so that the most significant k bits of the sampling counter will almost certainly represent m. Intuitively, the least significant l bits of M "absorb" the error. This will occur if $m2^l \leq M < (m + 1)2^l$. Note that $m < 2^k$. Let $\varepsilon = \frac{1}{2m}$. Then the Chernoff

bound [26, Theorems 4.4/4.5, part 2] and the union bound tell us that

$$\begin{aligned}
& \Pr\left[M \geq (m+1)2^l \text{ or } M < m2^l\right] \\
= \;& \Pr\left[M \geq (1+\varepsilon)\mathrm{E}[M] \text{ or } M < (1-\varepsilon)\mathrm{E}[M]\right] \\
\leq \;& e^{-\mathrm{E}[M]\varepsilon^2/3} + e^{-\mathrm{E}[M]\varepsilon^2/2} < e^{-m2^l\left(\frac{1}{2m}\right)^2/3} + e^{-m2^l\left(\frac{1}{2m}\right)^2/2} \\
= \;& e^{-\frac{2^{l-2}}{3m}} + e^{-\frac{2^{l-2}}{2m}} \qquad\qquad < e^{-2^{l-k-2}/3} + e^{-2^{l-k-2}/2}
\end{aligned}$$

Let $c \in \mathbb{N}$ be a constant. By setting $l = k+c$, the probability of error decreases exponentially in c:

$$\Pr\left[M \geq (m+1)2^l \text{ or } M < m2^l\right] < e^{-2^{c-2}/3} + e^{-2^{c-2}/2} < 2 \cdot 0.717^{2^{c-2}}. \tag{4.3.4}$$

For instance, letting $c = 6$ bounds the left-hand side of (4.3.4) below 0.0052.

The number of samples is $2^a = 2^{2k+c} = O((2^k)^2)$. Since $m < 2^k$, integers m such that $m^2 \ll n$ can be "probably exactly computed" using much fewer than n Bernoulli trials, and can therefore be computed by a sampling line without exceeding the boundaries of an $n \times n$ square.

4.3.3 Computing n Exactly

We have shown how to compute a number m exactly using a sampling line of length $O(m^2)$ and width $O(\log m)$. To compute n, the dimensions of the square, we must compute m_1, m_2, and m_3, which are the numbers represented by the bits of the binary expansion of n at positions congruent to 1 mod 3, 2 mod 3, and 0 mod 3, respectively. To compute all three of these numbers, we embed two extra sampling counters into the double counter, in addition to the sampling counter described in Section 4.3.2, to create a quadruple counter. This requires 8 sampling tiles instead of 2, in order to represent each of the possible outcomes of conducting three simultaneous Bernoulli trials, each trial used for estimating one of m_1, m_2, or m_3.

Given $i \in \{1, 2, 3\}$, let $b_i \in \{0, 1\}$ denote the outcome of the i^{th} of three simultaneous Bernoulli trials, and let $p_i(b_i)$ denote the probability we would like to associate with that

outcome. As noted in Section 4.3.2.2, the values of the c_i's are given by $p_i(1) = \frac{m_i 2^l + 2^{l-1}}{2^a}$, and $p_i(0) = 1 - p_i(1)$.

Since each of the three simultaneous Bernoulli trials is independent, we can calculate the appropriate concentration of the tile representing the three outcomes by multiplying the three outcome probabilities together. Then the required concentration of the tile representing outcomes b_1, b_2, b_3 is given by $p_1(b_1) \cdot p_2(b_2) \cdot p_3(b_3)$.

Once the values m_1, m_2, and m_3 are computed, we must remove the c least significant (bottom) bits from the bottom of the primary counter. Since c is a constant depending only on δ, it can be encoded into the tile types. We must then remove the bottom half of the remaining bits.[8] At this point, the concatenation of the bits on the tiles represent the binary expansion of n. Rather than expand them out to use three times as many tiles, we simply translate each of them to an octal digit, giving the octal representation of n, with one octal digit per tile replacing the three bits per tile. Finally, this representation of n is rotated 90 degrees counter-clockwise, used as the initial value for a decrementing, upwards-growing, base-8 counter, and used to fill in an $n \times n$ square using the standard construction [32]. Rotating n to face up starts the counter $2k+2$ tiles from the bottom of the construction so far. Furthermore, testing whether the counter has counted below 0 requires counting once beyond 0, using 2 more rows than the starting value of the counter. Therefore, to ensure that exactly an $n \times n$ square is formed, the value $n - 2k - 4$, rather than n exactly, is programmed into the tile concentrations to serve as the start value of the upwards-growing counter. An outline of this construction is shown in Figure 4.4.

4.3.4 Choice of Parameters

We now derive the settings of various parameters required to achieve a desired success probability and derive lower bounds on n necessary to allow the space required by the construction. To ensure probability of failure at most δ, we pick r, the number of stages of stopping tiles that must attach before the primary counter is sent the stop signal, so that $2 \cdot 0.9421^r \leq \frac{\delta}{4}$ as

[8] Isolating the most significant half of the bits can be done using a tile set similar to the algorithm one might use to program a single-tape Turing machine to compute the function $0^{2n} \mapsto 0^n$.

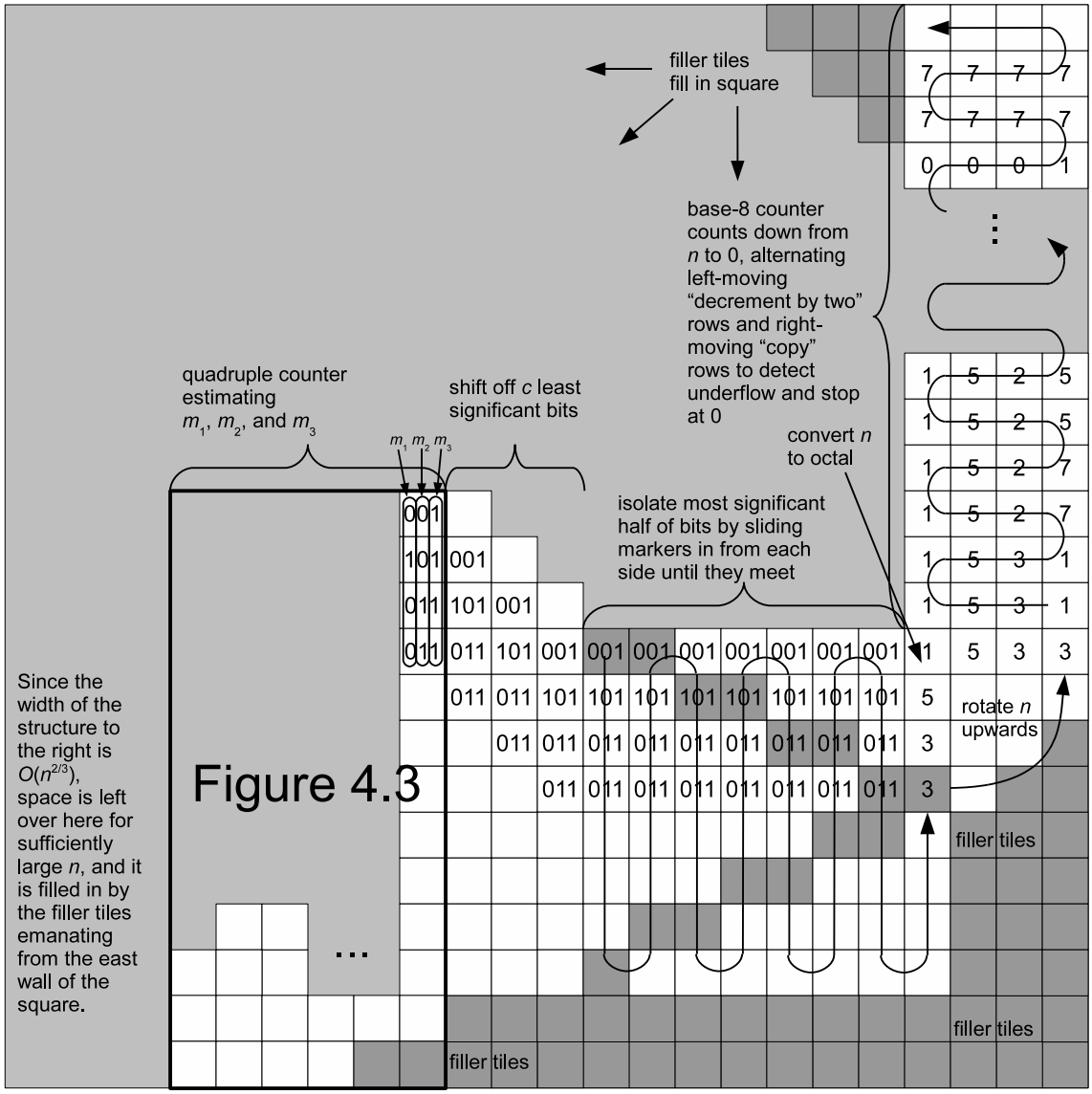


Figure 4.4: High-level overview of the entire construction, not at all to scale. For brevity, glue strengths and labels are not shown. The double counter number estimator of Figure 4.3 is embedded with two additional counters to create a quadruple counter estimating m_1, m_2, and m_3, shown as a box labeled as "Figure 4.3" in the above figure. In this example, $m_1 = 4$, $m_2 = 3$, and $m_3 = 15$, represented vertically in binary in the most significant 4 tiles at the end of the quadruple counter. Concatenating the bits of the tiles results in the string 001101011011, the binary representation of 859, which equals $n - 2k - 4$ for $n = 871$, so this example builds an 871 x 871 square. (Actually, 871 is too small to work with our construction, so the counter will exceed length 871, but we choose small numbers to illustrate the idea more clearly.) Once the counter ends, c tiles ($c = 3$ in this example) are shifted off the bottom, and the top half of the tiles are isolated ($k = 4$ in this example). Each remaining tile represents three bits of n, which are converted into octal digits, rotated to face upwards, and then used to initialize a base-8 counter that builds the east wall of the square. Filler tiles cover the remaining area of the square.

in (4.3.3):

$$r = \left\lceil \frac{\log \frac{\delta}{8}}{\log 0.9421} \right\rceil .$$

For example, choosing $r = 113$ achieves probability of error $\delta/4$ (in ensuring the counter stops between the numbers 2^{a-1} and 2^a) at most 0.0025.

To ensure that each of m_1, m_2, and m_3 are computed exactly, we set c, the number of extra bits used in the primary counter beyond $2k$, such that $e^{-2^{c-2}/3} + e^{-2^{c-2}/2} \leq \frac{\delta}{4}$, as in (4.3.4), or more simply, such that $2 \cdot 0.717^{2^{c-2}} \leq \frac{\delta}{4}$; i.e., set

$$c = 2 + \left\lceil \log \frac{\log \frac{\delta}{8}}{\log 0.717} \right\rceil .$$

For example, choosing $c = 7$ achieves probability of error $\delta/4$ (in ensuring that m_1 is computed correctly) at most 0.0025 (in fact, at most 0.000005).

By the union bound, the length of the sampling line and the values of m_1, m_2, and m_3 are computed with sufficient precision to compute the exact value of n with probability at least $1 - \delta$. The example values of r and c given above achieve $\delta \leq 0.01$.

The choices of r and c imply a lower bound on the value of n necessary to allow sufficient space to carry out the construction. Clearly the counter must reach at least value r, since there are r different stopping stages. The more influential factor will be the value c, which doubles the space necessary to run the counter each time it is incremented by 1. n requires $\lfloor \log n \rfloor + 1$ bits to represent, but our estimation will be a string of length the next highest multiple of 3 above $\lfloor \log n \rfloor + 1$. Therefore, each of m_1, m_2, and m_3 requires

$$k = \left\lceil \frac{\lfloor \log n \rfloor + 1}{3} \right\rceil$$

bits to represent. Recall that the primary counter will have height $2k + c$ and count to 2^{2k+c} (so long as $r \leq 2^{2k+c}$). Then, c columns are required to shift off the constant c bits from the least significant bits of the counter, and $2k$ columns are required to shift off the least significant half of the bits of the counter to isolate the k most significant bits. k columns are needed to

translate the groups of three bits into octal and to rotate this string to face upwards for the square-building counter.

Hence, the total length required along the bottom of the square to compute n is $\max\{r, 2^{2k+c}\}+c+3k$. Expanding out the definitions of r, k, and c derived above gives the lower bound $b(n, \delta)$ on n described in Theorem 4.3.1.

For sufficiently large n and small enough δ, r is much smaller than 2^{2k+c}, so the latter term dominates. For example, to achieve probability of error $\delta \leq 0.01$ requires $n >$ 8,000,000. According to preliminary experimental tests, in practice, a smaller value of c is required than the theoretical bounds we have derived. For example, if the desired error probability is $\delta = 0.01$, setting $c = 7$ satisfies the analysis given above, but in experimental simulation, $c = 3$ appears to suffice for probability of error at most 0.01, and reduces the space requirements by a factor of $2^{7-3} = 16$. In this case, $n = 9000$ can be computed by a construction that will stay within the 9000 x 9000 square.

A simulated implementation of this tile assembly system using the ISU TAS Tile Assembly Simulator [27] is available at `http://www.cs.iastate.edu/~lnsa/software.html`. The tile set uses approximately $4500 + 9c + 4r$ tile types, where r and c are calculated from δ as above.

4.4 Conclusion

We have described how a single tile set in Winfree's abstract tile assembly model, appropriately "programmed" by setting tile concentrations, exactly assembles an $n \times n$ square with high probability, for any sufficiently large n.

4.4.1 Future Work

The focus of the present chapter is on conceptual clarity. We have therefore described the simplest (i.e., easiest to understand, but not necessarily smallest) version of the tile assembly system that achieves the desired *asymptotic* result that an $n \times n$ square assembles with high probability for sufficiently large n. We now observe that this theoretical result could be improved in practice by complicating the tile set.

Our implementation of the tile set uses approximately $4500 + 9c + 4r$ tiles, where, for example, $r = 113$ and $c = 7$ are sufficient to achieve error probability $\delta \leq 0.01$. The tiles are so numerous because of the need to simultaneously represent 4 bits in a tile, in addition to information such as the significance of the bit (MSB, LSB, or interior bit), and doing computation such as addition, which requires tiles that can handle the 2^8 possible input bit + carry signals. Putting together a few such modules of tile sets results in thousands of tiles before too long. The number of tile types could be reduced by splitting the estimation of m_1, m_2, and m_3 into three distinct geometrical regions, so that each tile is required to remember less information. This would complicate the tile set, as it would require more shifting tricks to ensure sufficient room for all counters, and would require bringing the bits back together again at the end, but it would likely reduce the number of tile types.

A large value of n is required to achieve a probability of success at least $(1-\delta)$ for reasonably small δ; $n > 8 \cdot 10^6$ is required to estimate n with 99% chance of correctness (in theory, but apparently not in practice, as discussed in Section 4.3.4). This shortcoming can be compensated in a number of ways.

In a similar spirit to the linear speedup theorem, more than three simultaneous Bernoulli trials may be conducted with each sampling tile. For example, conducting 6 Bernoulli trials with each sampling tile would estimate two bits of $m_1, \ldots, m_3$ with per sampling tile, rather than one bit, halving the required length of the sampling line. This would result in a prohibitively large tile set, however; as the number of tile types increases exponentially with the number of simultaneous Bernoulli trials per tile type.

A conceptually simpler and practically more feasible improvement is to use 0/1-valued tile concentrations to simulate tile *type* programming (i.e., designing tile types specially to build a particular size square, as in [32]) for small values of n, by including tile types that deterministically construct an $n \times n$ square for each small n, setting concentrations of those tiles to be 1 and setting concentrations of all other tiles to be 0. Many of the same square-building tile types for can be reused for different values of n (see [32]), with the different values of n largely being dependent on $\approx \log n$ hard-coded tiles that immediately attach to the seed.

For singly-seeded tile systems, $3 \log N + O(1)$ tiles are required to handle all $n \leq N$: for each $i \in \{1, \ldots, \log N\}$ that represents the position of a bit of n, three tile types are required, one representing "0 at position i", one representing "1 at position i" (each of which has double-strength bonds on two sides), and one representing "end of string at position i" (with a double-strength bond on one side and a zero-strength bond on the other). Though this solution lacks the "feel" of tile concentration programming, it is likely that real-life implementations of tile concentration programming will need to use such hard-coding tricks for smaller structures that lack the space to carry out the amount of sampling required to reconstruct precise inputs solely from tile concentrations.

Improved analysis of the tail bounds could reduce the number of samples needed to guarantee exact estimation of the numbers. Preliminary experimental analysis suggests that these bounds are not tight.

An alternate improvement to the tile set would be to combine the present technique with the Kao-Schweller technique of building a sampling line inside of a square, to more efficiently use the n^2 space available to carry out the estimation. However, square-building is not necessarily the only application of this technique, as discussed next.

The primary novel contribution of this chapter is a tile set that, through appropriate tile concentration programming, forms a thin structure of length $O(n^{2/3})$ and height $O(\log n)$,[9] whose rightmost tiles encode the value of n. The number n could be used to assemble useful structures other than squares. For the task of building a square, this construction wastes the $\approx n^2$ space available above the thin rectangle, but for computing other structures, it may be advantageous that the rectangle is kept thin. For instance, biochemists routinely use filters (e.g., Millipore Ultrafiltration Membranes) and porous resins [7] to separate proteins based on size, in order to isolate one particular protein for study. The ability to precisely control the size of the filter holes or resin beads would allow for more targeted filtering of proteins than is possible at the present time. DNA is likely too reactive with amino acids to be used as the

[9] By partitioning n's binary representation into t rather than three subsequences, for $t \in \mathbb{N}$ a constant, the number of trials needed to estimate n is $O(n^{2/t})$. However, the constant factors in the $O()$ increase, making the technique even less feasible for small values of n. But if some application requires an asymptotically very short line, the line can be made length $O(n^{\varepsilon})$ for any $\varepsilon > 0$ using this technique.

substrate for such a structure, so an implementation of the tile assembly model not based on DNA would be required for such a technique.

Similarly, polyacrylamide gel electrophoresis [36], another technique for discriminating biological molecules on the basis of size, requires molecular mass size markers, which are control molecules of known molecular mass, in order to compare against the molecule of interest on the gel. At the present time, some naturally-occurring molecules of known mass are used, but their masses are not controllable, and the ability to quickly and easily assemble molecules of precisely a desired target mass would be useful in experiments requiring mass markers that differ from the standards. Again, DNA is a special case in which this idea is unnecessary, since precise standards have been developed for DNA gels (e.g., Novagen DNA Markers). But the tile assembly model may one day be implemented using substances that are appropriate for a protein gel.

Chapter 5. A Domain-Specific Language for Programming in the Tile Assembly Model

This chapter is joint work with Matt Patitz and was originally published as [14].

5.1 Introduction

5.1.1 Background

The current practice of the design of tile types for the aTAM and related models is not unlike early machine-level programming. Numerous theoretical papers [18, 32, 37, 38] focus on the *tile complexity* of systems, the minimum number of tile types required to assemble certain structures such as squares. A major motivation of such complexity measures is the immense level of laboratory effort presently required to create physical implementations of tiles.

Early electronic computers occupied entire rooms to obtain a fraction of the memory and processing power of today's cheapest mobile telephones. This limitation did not stop algorithm developers from creating algorithms, such as divide-and-conquer sorting and the simplex method, that find their most useful niche when executed on data sets that would not have fit on all the memory existing in the world in 1950. Similarly, we hope and expect that the basic components of self-assembly will one day, through the ingenuity of physical scientists, become cheap and easy to produce, not only in quantity but in variety. The *computational* scientists will then be charged with developing disciplined methods of organizing such components so that systems of high complexity can be designed without overwhelming the engineers producing the design. In this chapter, we introduce a preliminary attempt at such a disciplined method of controlling the complexity of designing tile assembly systems in the aTAM.

Simulated tile assembly systems of moderate complexity cannot be produced by hand; one

must write a computer program that handles the drudgery of looping over related groups of individual tile types. However, even writing a program to produce tile types directly is error-prone, and more low-level than the ways that we tend to think about tile assembly systems.

Fowler [16] suggests that a *domain-specific language (DSL)* is an appropriate tool to introduce into a programming project when the syntax or expressive capabilities of a general-purpose programming language are awkward or inadequate for certain portions of the project. He distinguishes between an *external DSL*, which is a completely new language designed especially for some task (such as SQL for querying or updating databases), and an *internal DSL*, which is a way of "hijacking" the syntax of an existing general-purpose language to express concepts specific to the domain (e.g. Ruby on Rails for writing web applications). An external DSL may be as simple as an XML configuration file, and an internal DSL may be as simple as a class library. In either case the major objective is to express "commands" in the DSL that more closely model the way one thinks about the domain than the "host" language in which the project is written.

If the syntax and semantics of the DSL are precisely defined, this facilitates the creation of a *semantic editor* (text-based or visual), in which programs in the DSL may be produced using a tool that can visually show semantically meaningful information such as compilation or even logical errors, and can help the programmer directly edit the abstract components of the DSL, instead of editing the source code directly. For example, Intellij IDEA and Eclipse are two programs that help Java programmers to do refactorings such as variable renaming or method inlining, which are at their core operations that act directly on the abstract syntax tree of the Java program rather than on the text constituting the source code of the program. We have kept such a tool in mind as a guide for how to appropriately structure the DSL for designing tile systems.

We structure this chapter in such a way as to de-emphasize any dependence of the DSL on Python in particular or even on object-oriented class libraries in general. The DSL provides a high-level way of *thinking* about tile assembly programming, which, like any high-level language or other advance in software engineering, not only automates mundane tasks better left to

computers, but also *restricts* the programmer from certain error-prone tasks, in order to better guide the design, following the dictum of Antoine de Saint-Exupery that a design is perfected "not when there is nothing left to add, but nothing left to take away."

5.1.2 Brief Outline of the DSL

We now briefly outline the design of the DSL. Section 5.2 provides more detail.

The most fundamental component of designing a tile assembly system manually is the tile type. In our DSL, the most fundamental component is an object known as a *tile template*. A tile template represents a group of tile types (each tile type being an *instance* of the tile template), sharing the same input sides, output sides, and function that transforms input signals into output signals. The two fundamental operations of the DSL are *join* and *add transition*. Both make the assumption that each tile template has well-defined input and output sides. In a join, an input tile template A is connected to an output tile template B in a certain direction $d \in \{N, S, E, W\}$, expressing that an instance t_A of A may have on its output side in direction d an instance t_B of B. This expresses an intention that in the growth of the assembly, t_A will be placed first, then t_B, and they will bind with positive strength, with t_A passing information to t_B. Whereas a join is an operation connecting the output side of a tile template to the input side of another, a transition "connects" input sides to output sides within a single tile template, by specifying how to compute information on the output side as a function of information on the input sides. This is called *adding* a transition rather than *setting*, since the information on the output sides may contain multiple independent output signals, and their computations may be specified independently of one another if convenient.

This notion of independent signals is modeled already in other DSLs for the aTAM [4] and for similar systems such as cellular automata [10]. The join operation, however, makes sense in the aTAM but not in a system such as a cellular automaton, where each cell contains the same transition function(s). The notion of tile templates allows one to break an "algorithm" for assembly into separate "stages", each stage corresponding to a different tile template, with each stage being modularized and decoupled from other stages, except through well-defined

and restricted signals passed through a join. There is a rough analogy with lines of code in a program: a single line of code may execute more than once, each time executed with the state of memory being different than the previous. Similarly, many different tile types, with different actual signal values, generated from the same tile template, may be placed during the growth of an assembly. Another difference between our language and that of [4] is that our language appears to be more general; rather than being geared specifically toward the creation of geometric shapes, our language is more low-level, but also more general.[1] Additionally, [39] presents a DSL which provides a higher level framework for the modeling of various self-assembling systems, while the focus of our DSL on the aTAM creates a more powerful platform for the development of complex tile assembly systems.

This chapter is organized as follows. Section 5.2 describes the DSL for tile assembly programming in more detail and gives examples of design and use. Section 5.3 concludes the chapter and discusses future work and open theoretical questions.

A preliminary implementation of the DSL and the visual editor can be found at `http://www.cs.iastate.edu/~lnsa`.

5.2 Description of Language

The DSL is written as a class library in the Python programming language. It is designed as a set of classes which encapsulate the logical components needed to design a tile assembly system in the aTAM where the temperature value $\tau = 2$.

The DSL is designed around the principal notion that data moves through an assembly as 'signals' which pass through connected tiles as the information encoded in the input glues, allowing a particular tile type to bind in a location, and then, based on the 'computation' performed by that tile type, as the resultant information encoded in the glues of its output edges. (Of course, tiles in the aTAM are static objects so the computation performed by a given tile type is a simple mapping of one set of input glues to one set of output glues which is hardcoded at the time the tile type is created.) Viewed in this way, signals can be seen

[1] An extremely crude analogy would be that the progression *manual tile programming* → *Doty-Patitz* → *Becker* is roughly analogous to *machine code* → *C* → *Logo*.

to propagate as tiles attach and an assembly forms, and it is these signals which dictate the final shape of the assembly. Using this understanding of tile-based self-assembly, we designed the DSL from the standpoint of building tile assembly systems around the transmission and processing of such signals.

We present a detailed overview of the objects and operations provided by the DSL, then demonstrate an example exhibiting the way in which the DSL is used to design a tile set.

5.2.1 Client-side description

The DSL provides a collection of objects which represent the basic design units and a set of operations which can be performed on them. We describe these objects and operations in this section, without describing the underlying data structures and algorithms implementing them.

The DSL strictly enforces the notion of input and output sides for tile types, meaning that any given side can be designated as receiving signals (an input side), sending signals (an output side), or neither (a blank side).

5.2.1.1 DSL objects

Tile system The highest level object is the *tile system* object, which represents the full specification of a temperature 2 tile assembly system. It contains child objects representing the tile set and seed specification, and provides the functionality to write them out to files in a format readable by the ISU TAS simulator [27] (`http://www.cs.iastate.edu/~lnsa/software.html`).

Tile In some cases, especially for tiles contained within seed assemblies, there is no computation being performed. In such situations, it may be easier for the programmer to fully specify the glues and properties of a tile type. The *tile* object can be used to easily create such hardcoded tile types.

Tile template The principle design unit in the DSL is the tile template. This object represents a collection of tile types which share the following properties:

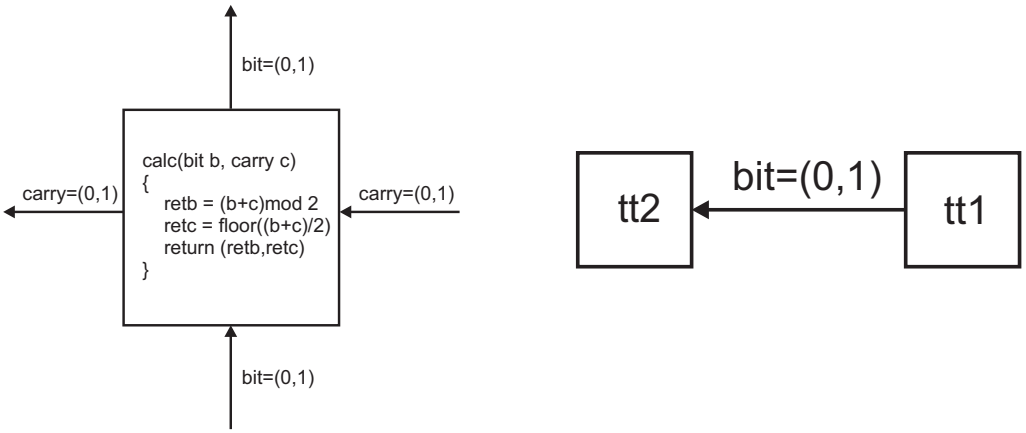
- They have exactly the same input, output, and blank sides.
- The types of signals received/transmitted on every corresponding input/output side are identical.
- The computation performed to transform the input signals to output signals can be performed by the same function, using the values specific to each instantiated tile type.

Logically, a tile template represents the set of tile types which perform the same computation on different values of the same input signal types.

Tile set template A *tile set template* contains all of the information needed to generate a tile set. It contains the sets of tile and tile template objects included in the tile set, as well as the logic for performing joins, doing error checking, etc. The tile set template object encapsulates information and operations that require communication between more than one tile template object, such as the *join* operation, whereas a tile template is responsible for operations that require information entirely local to the tile template, such as *add transition*.

Signal A *signal* is simply the name of a variable and the set of allowable values for it. For example, a signal used to represent a binary digit could be defined by giving it the name *bit* and the allowable values 0 and 1.

Transition *Transitions* are the objects which provide the computational abilities of tile templates. A transition is defined as a set of input signal names, output signal names, and a function which operates on the input signals to yield output signals. The logic of a function can be specified as a table enumerating all input signals and their corresponding outputs, a Python expression, or a Python function, yielding the full power of a general purpose programming language for performing the computations. An example is shown in Figure 5.1a.



(a) Transition: The function *calc* takes bit/carry values as input from the bottom/right, and computes new bit/carry values which are then output on the top/left.

(b) Join: The tile template on the right (tt1) is sending the signal *bit* with allowable values $(0, 1)$ to the tile template on the left (tt2).

Figure 5.1: Logical representations of some DSL objects

5.2.1.2 DSL operations

Join The primary operation between tile templates, which defines the signals that are passed and the paths that they take through an assembly, is called a *join*. Joins are performed between the complementary sides (north and south, or east and west) of two tile templates (which need not be unequal), or a tile and a tile template. If one parameter is a tile, then all signals must be given just one value; otherwise a set of possible values that could be passed between the tile templates is specified (which can still be just one). A join is directional, specifying the direction in which a signal (or set of signals) moves. This direction defines the input and output sides of the tile templates which a join connects. An example is shown in Figure 5.1b.

Add transition Transition objects can be added to tile template objects, and indicate how to compute the output signals of a tile template from the input signals. If convenient, a single transition can compute more than one output signal by returning a tuple. Each output signal of a tile template with more than one possible value must have a transition computing

it.

Add chooser There may be multiple tile templates that share the same input signal values on all of their input sides. Depending on the join structure, the library may be able to use annotations (see below) to avoid "collisions" resulting from this, but if the tile templates share joins, then it may not be possible (or desirable) for the library to automatically calculate which tile template should be used to create a tile matching the inputs. In this case, a user-defined function called a *chooser* must be added so that the DSL can ensure that only a single tile type is generated for each combination of input values. This helps to avoid accidental nondeterminism.

Set property Additional properties such as the display string, tile color, and tile type concentrations can be set using user-defined functions for each property.

5.2.2 Additional features

The DSL provides a number of additional, useful features to programmers, a few of which will be described in this section. First, the DSL performs an analysis of the connection structure formed between tile templates by joins and creates *annotations*, or additional information in the form of strings, which are appended to glue labels. These annotations ensure that only tile types created from tile templates which are connected by joins can bind with each other in the directions specified by those joins, preventing the common error of accidentally allowing two tiles to bind that should not because they happen to communicate the same information in the same direction as two unrelated tiles.

Although some forms of 'accidental' nondeterminism are prevented by the DSL, it does provide methods by which a programmer can intentionally create nondeterminism. Specifically, a programmer can either design a chooser function which returns more than one output tile type for a given set of inputs, or one can add *auxiliary inputs* to a tile template, which are input signals that do not come from any direction.

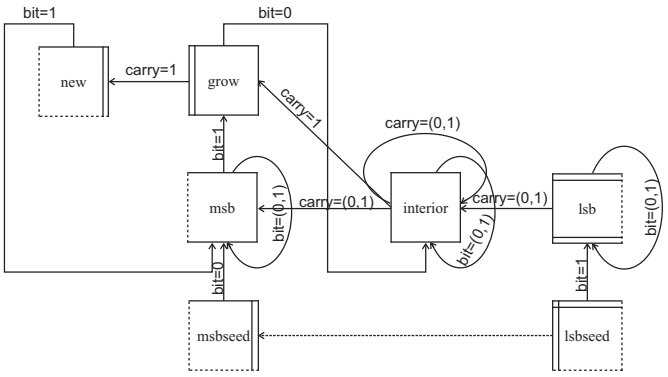
The DSL provides additional error-checking functionality. For example, each tile template must have either one strength-2 input or two strength-1 inputs. As another example, the programmer is forced to specify a chooser function if the DSL cannot automatically determine a unique output tile template, which is a common source of accidental nondeterminism in tile set design.

5.2.3 Example construction

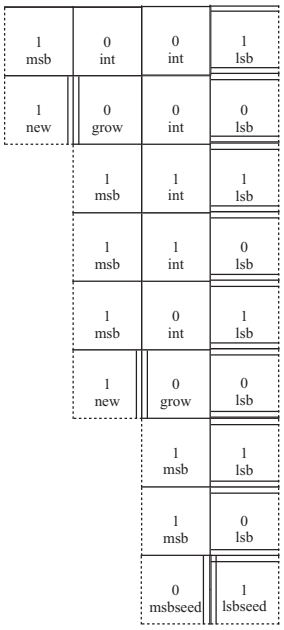
In this section, we present an example of how to use the DSL to produce a tile assembly system which assembles a log-width binary counter. In order to slightly simplify this example, the seed row of the assembly, representing the value 1, will be two tiles wide instead of one. All other rows will be of the correct width, i.e. a row representing the value n will be $\lceil \log_2(n+1) \rceil$ tiles wide.

Figure 5.2a shows a schematic diagram representing a set of DSL objects, namely tiles, tile templates and joins, which can generate the necessary tile types. It is similar in appearance to how such a diagram appears in the visual editor. The squares labeled *lsbseed* and *msbseed* represent hard-coded tiles for the seed row. The other squares represent tile templates, with the names shown in the middle. The connecting lines represent joins, which each have a direction specified by a terminal arrow and a signal which is named (either *bit* or *carry*) and has a range of allowable values (either 0, or 1, or both). The dashed line between *lsbseed* and *msbseed* represents an implicit join between these two hard-coded tile types because they were manually assigned the same glues.

In addition to the joins, transition functions must be added to the *lsb*, *interior*, and *msb* tile templates to compute their respective output signals, since each has multiple possible values for those signals (whereas *grow*, for instance, which always outputs carry=1 to its west and bit=0 to its north, so needs no transitions). Finally, since the tile templates *msb* and *grow* have overlapping input signal values of bit=1 from *msb* on the south and carry=1 from *interior* on the east, a chooser function must be supplied to indicate that an instance of *grow* should be created for the signals bit=1 and carry=1, and an instance of *msb* should be created in the



(a) Schematic diagram depicting the tile templates and joins which are used to generate a tile set that self-assembles a log-width binary counter. All east-west joins pass the *carry* signal and south-north joins pass the *bit* signal (although some of them are restricted to subsets of the allowable values $(0, 1)$). Transitions are not shown explicitly, but, for example, the tile template *interior* would have a transition as shown in Figure 5.1a.



(b) First 9 rows of the assembly of a log-width binary counter. Each tile is labeled with its bit value on top and the name of the tile template from which it was generated on the bottom.

Figure 5.2: Schematic diagram and partial assembly of generated tile types for log-width counter.

other three circumstances. Figure 5.2b shows a partial assembly of tiles generated from the tile template diagram of Figure 5.2a, without glue labels shown explicitly.

5.3 Conclusion and Future Work

We have described a domain-specific language (DSL) for creating tile sets in the abstract Tile Assembly Model. This language is currently implemented as a Python class library but is framed as a DSL to emphasize its role as a high-level, disciplined way of thinking about the creation of tile systems.

5.3.1 Semantic Visual Editor

The DSL is implemented as a Python class library, but one thinks of the "real" programming as the creation of visual tile templates and the joins between them, as well as the addition of signal transitions. We have implemented a visual editor that removes the Python programming and allows the direct creation of the DSL objects and execution of its operations. It detects errors, such as a tile template having only one input side of strength 1, while temporarily allowing the editing to continue. Other types of errors, that do not aid the user in being allowed to persist, such as the usage of a single side as both an input and output, are prohibited outright. Many improvements in error detection and handling remain to be implemented and support remains to be added for several features of the DSL.

One can therefore create a tile set by directly drawing the tile templates as shown in Figure 5.2a, while receiving helpful tips from a semantically-aware editor about errors.

5.3.2 Avoidance of Accidental Nondeterminism

Initially, our hope had been to design a DSL with the property that it could be used in a straightforward way to design a wide range of existing tile assembly systems, but was sufficiently restricted that it could be guaranteed to produce a deterministic TAS, so long as nondeterminism was not directly and intentionally introduced through chooser functions or auxiliary inputs. Unfortunately, it is straightforward to use the DSL to design a TAS that

begins the parallel simulation of two copies of a Turing machine so that if (and only if) the Turing machine halts, two "lines" of tiles are sent growing towards each other to compete nondeterministically, whence the detection of such nondeterminism is undecidable. A common cause of "accidental nondeterminism" in the design of tile assembly systems is the use of a single side of a tile type as both input and output; this sort of error is prevented by the nature of the DSL in assigning each tile template unique, unchanging sets of input sides and output sides. However, we cannot see an elegant way to restrict the DSL further to automatically prevent or statically detect the presence of the sort of "geometric nondeterminism" described in the Turing machine example. The development of such a technique would prove a boon to the designers of tile assembly systems. Conversely, consider Blum's result [6] that any programming language in which all programs are guaranteed to halt requires uncomputably large programs to compute some functions that are trivially computable in a general-purpose language. Perhaps an analogous theorem implies that any restriction statically enforcing determinism also cripples the Tile Assembly Model, so that accidental nondeterminism in tile assembly, like accidental infinite loops in software engineering, would be an unavoidable fact of life, and time is better spent developing formal methods for proving determinism rather than hoping that it could be automatically guaranteed.

5.3.3 Further Abstractions

To better handle very large and complex constructions, it would be useful to support further abstractions, such as grouping sets of tile templates and related joins into "modules" which could then be treated as atomic units. These modules should have well-defined interfaces which allow them to be configured and combined to create more complicated, composite tile sets.

5.3.4 Other Self-Assembly Models

The abstract Tile Assembly Model is a simple but powerful tool for exploring the theoretical capabilities and limitations of molecular self-assembly. However, it is an (intentionally) overly-simplified model. Generalizations of the aTAM, such as the kinetic Tile Assembly Model

[46,47], and alternative models, such as graph self-assembly [30], have been studied theoretically and implemented practically. We hope to leverage the lessons learned from designing the aTAM DSL to guide the design of more advanced DSLs for high-level programming in these alternative models.

Chapter 6. Limitations of Self-Assembly at Temperature One

This chapter is joint work with Matt Patitz and Scott Summers and was originally published as [15].

6.1 Introduction

Despite its deliberate over-simplification, the TAM is a computationally and geometrically expressive model at temperature 2. The reason is that, at temperature 2, certain tiles are not permitted to bond until *two* tiles are already present to match the glues on the bonding sides, which enables cooperation between different tile types to control the placement of new tiles. Winfree [45] proved that at temperature 2 the TAM is computationally universal and thus can be directed algorithmically.

In contrast, we aim to prove that the lack of cooperation at temperature 1 inhibits the sort of complex behavior possible at temperature 2. Our main theorem concerns the *weak self-assembly* of subsets of $\mathbb{Z}^2$ using temperature 1 tile assembly systems. Informally, a set $X \subseteq \mathbb{Z}^2$ weakly self-assembles in a tile assembly system $\mathcal{T}$ if some of the tile types of $\mathcal{T}$ are painted black, and $\mathcal{T}$ is guaranteed to self-assemble into an assembly α of tiles such that X is precisely the set of integer lattice points at which α contains black tile types. As an example, Winfree [45] exhibited a temperature 2 tile assembly system, directed by a clever XOR-like algorithm, that weakly self-assembles a well-known set, the discrete Sierpinski triangle, onto the first quadrant. Note that the underlying *shape* (set of all points containing a tile, whether black or not) of Winfree's construction is an infinite canvas that covers the entire first quadrant, onto which a more sophisticated set, the discrete Sierpinski triangle, is painted.

We show that under a plausible assumption, temperature 1 tile systems weakly self-assemble

only a limited class of sets. To prove our main result, we require the hypothesis that the tile system is *pumpable*. Informally, this means that every sufficiently long path of tiles in an assembly of this system contains a segment in which the same tile type repeats (a condition clearly implied by the pigeonhole principle), and that furthermore, the subpath between these two occurrences can be repeated indefinitely ("pumped") along the same direction as the first occurrence of the segment, without "colliding" with a previous portion of the path. We give a counterexample in Section 6.2 (Figure 6.1) of a path in which the same tile type appears twice, yet the segment between the appearances cannot be pumped without eventually resulting in a collision that prevents additional pumping. The hypothesis of pumpability states (roughly) that in every sufficiently long path, despite the presence of some repeating tiles that cannot be pumped, *there exists* a segment in which the same tile type repeats that *can* be pumped. In the above-mentioned counterexample, the paths constructed to create a blocked segment always contain some previous segment that *is* pumpable. We conjecture that this phenomenon, pumpability, occurs in every temperature 1 tile assembly system that produces a unique infinite structure. We discuss this conjecture in greater detail in Section 4.4. We also prove that if the terminal assembly of a directed tile assembly system has no glue mismatches between neighboring tiles, then the tile assembly system is pumpable.

A *linear* set $X \subseteq \mathbb{Z}^2$ is a set of integer lattice points with the property that there are three vectors $\vec{b}$ (the "base point" of the set), $\vec{u}$, and $\vec{v}$ (the two periods of the set), such that $X = \left\{ \vec{b} + n \cdot \vec{u} + m \cdot \vec{v} \;\middle|\; n, m \in \mathbb{N} \right\}$. That is, a linear set is a "two-dimensionally periodic" set that repeats infinitely along two vectors (linearly independent vectors in the non-degenerate case), starting at some base point $\vec{b}$. We show that any directed, pumpable, temperature 1 tile assembly system weakly self-assembles a set $X \subseteq \mathbb{Z}^2$ that is a finite union of linear sets, known also as a *semilinear* set. Semilinear sets are computationally very simple. They have been shown, for example, to be equivalent to those sets definable in the (very weak) theory of Presburger arithmetic [29,40], the first-order theory of natural numbers with only the addition operation (but lacking multiplication). They have also been shown [17] to characterize the set of languages (of tuples of natural numbers) generated by *reversal-bounded counter machines*,

which are finite-state machines without an input tape, and a finite number (two in our case since we work in $\mathbb{Z}^2$) of counters – a register containing a natural number that can be incremented, decremented, or tested for 0 – with the property that each counter can only switch between incrementing and decrementing a bounded number of times. For completeness, we give a self-contained proof (Observation 6.3.2) that the projection of any semilinear set onto the x-axis, when expressed in unary, is a regular language.

It is our contention that weak self-assembly captures the intuitive notion of what it means to "compute" with a tile assembly system. For example, the use of tile assembly systems to build shapes is captured by requiring all tile types to be black, in order to ask what set of integer lattice points contain any tile at all (so-called *strict self-assembly*). However, weak self-assembly is a more general notion. For example, Winfree's above mentioned result shows that the discrete Sierpinski triangle weakly self-assembles at temperature 2 [33], yet this shape does not strictly self-assemble at *any* temperature [22]. Hence weak self-assembly allows for a more relaxed notion of set building, in which intermediate space can be used for computation, without requiring that the space filled to carry out the computation also represent the final result of the computation.

As another example, there is a standard construction [45] by which a single-tape Turing machine may be simulated by a temperature 2 tile assembly system. Regardless of the semantics of the Turing machine (whether it decides a language, enumerates a language, computes a function, etc.), it is routine to represent the result of the computation by the weak self-assembly of some set. For example, Patitz and Summers [28] showed that for any decidable language $A \subseteq \mathbb{N}$, A's projection along the X-axis $\left(\text{the set } \left\{ \ (x,0) \in \mathbb{N}^2 \ \middle| \ x \in A \ \right\}\right)$ weakly self-assembles in a temperature 2 tile assembly system. As another example, if a Turing machine computes a function $f : \mathbb{N} \to \mathbb{N}$, it is routine to design a tile assembly system based on Winfree's construction such that, if the seed assembly is used to encode the binary representation of a number

$n \in \mathbb{N}$, then the tile assembly system weakly self-assembles the set

$$\left\{ (k, 0) \in \mathbb{N}^2 \;\middle|\; \begin{array}{l} \text{the } k^{\text{th}} \text{ least significant bit of the} \\ \text{binary representation of } f(n) \text{ is 1} \end{array} \right\}.$$

Our result is motivated by the thesis that if a tile assembly system can reasonably be said to "compute", then the result of this computation can be represented in a straightforward manner as a set $X \subseteq \mathbb{Z}^2$ that weak self-assembles in the tile assembly system, or a closely related tile assembly system. Our examples above provide evidence for this thesis, although it is as informal and unprovable as the Church-Turing thesis. On the basis of this claim, and the triviality of semilinear sets (shown more formally in Observation 6.3.2), we conclude that our main result implies that directed, pumpable, temperature 1 tile assembly systems are incapable of general-purpose deterministic computation, without further relaxing the model.

This chapter is organized as follows. Section 6.2 introduces new definitions and concepts that are required in proving our main theorem. Section 6.3 states our main theorem, and contains an observation with a self-contained proof justifying the suggestion that semilinear sets are computationally very simple, based on their relationship to regular languages. Section 6.4 shows an application of our theorem, showing that, unlike the case of temperature 2, no non-trivial discrete self-similar fractal – such as the discrete Sierpinski triangle – weakly self-assembles at temperature 1 in a directed, pumpable tile assembly system. Section 6.5 concludes the chapter and discusses open questions. Section 6.5 also observes that a relaxation of the tile assembly model, allowing negative glue strengths and allowing glues with different colors to interact, *is* capable of general-purpose computation. Section 6.6 is a technical appendix containing proofs of some results.

6.2 Pumpability, Finite Closures, and Combs

Throughout this section, let $\mathcal{T} = (T, \sigma, 1)$ be a directed TAS, and α be the unique assembly satisfying $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$. Further, we assume without loss of generality that σ consists of a single "seed" tile type placed at the origin.

Given, $\vec{0} \neq \vec{v} \in \mathbb{Z}^2$, a *$\vec{v}$-semi-periodic path in α originating at $\vec{a}_0 \in \text{dom } \alpha$* is an infinite, simple path $\pi = (\vec{a}_0, \vec{a}_1, \ldots)$ in the binding graph G_α such that there is a constant $k \in \mathbb{N}$ such that, for all $j \in \mathbb{N}$, $\pi[j+k] = \pi[j] + \vec{v}$, and $\alpha(\pi[j+k]) = \alpha(\pi[j])$. Intuitively, $\vec{v}$ is the "geometric" period of the path – the straight-line vector between two repeating tile types – while k is the "linear" period – the number of tiles that must be traversed along the path before the tile types repeat, which is at least $||\vec{v}||_1$, but possibly larger if the segment from $\pi[j]$ to $\pi[j] + \vec{v}$ is "winding".

An *eventually $\vec{v}$-semi-periodic path in α originating at $\vec{a}_0 \in \text{dom } \alpha$* is an infinite, simple path $\pi = (\vec{a}_0, \vec{a}_1, \ldots)$ in the binding graph G_α for which there exists $s \in \mathbb{N}$ such that the path $\pi' = (\pi[s], \pi[s+1], \ldots)$ is a v-semi-periodic path in α originating at $\pi[s]$. Let the *initial segment length* be the smallest index s^* such that $\pi' = (\pi[s^* - k], \pi[s^* - k + 1], \ldots)$ is a $\vec{v}$-semi-periodic path originating at the point $\pi[s^* - k]$. The *initial segment of π* is the path $\pi[0 \ldots i^* - 1]$ (for technical reasons, we enforce the initial segment of π to contain the simple path $\pi[0 \ldots s]$ along with one period of π'). The *tail of π* is $\pi[s^* \ldots]$. Note that the tail of an eventually $\vec{v}$-semi-periodic path is simply a $\vec{v}$-semi-periodic path originating at $\pi[s^*]$. An *eventually $\vec{v}$-periodic path in α* is an eventually $\vec{v}$-periodic path in α originating at $\vec{a}_0$ for some $\vec{a}_0 \in \text{dom } \alpha$. We say that π is a *$\vec{v}$-periodic* path in α if $\pi = (\ldots, \vec{a}_{-1}, \vec{a}_0, \vec{a}_1, \ldots)$ is a two-way infinite simple path such that, for all $j \in \mathbb{Z}$, $\alpha(\pi[j] + \vec{v}) = \alpha(\pi[j])$.

Let $\vec{w}, \vec{x} \in \text{dom } \alpha$, π be a simple path from $\vec{w}$ to $\vec{x}$ in the binding graph G_α, and $i, k \in \mathbb{N}$ with $0 \leq i < k \leq |\pi|$. We say that π has a *pumpable segment* $\pi[i \ldots k]$ (with respect to $\vec{v} = \pi[k] - \pi[i]$) if there exists a $\vec{v}$-semi-periodic path π' in α originating at $\pi[i]$ and $\pi'[0 \ldots k - i] = \pi[i \ldots k]$.

Intuitively, the path π has a pumpable segment if, after some initial sequence of tile types, it consists of a subsequence of tile types which is repeated in the same direction an infinite number of times, one after another. Figure 6.1 shows an assembly in which the same tile type repeats along a path, but the segment between the occurrences is not pumpable.

Let $c \in \mathbb{N}$ and $\vec{v} \in \mathbb{Z}^2$. The *diamond of radius c centered about the point $\vec{v}$* is the set of points defined as $D(c, \vec{v}) = \{\ (x, y) + \vec{v} \mid\ |x| + |y| \leq c\ \}$. Let $\vec{w}, \vec{x} \in \text{dom } \alpha$, $c \in \mathbb{N}$, and π be a simple path from $\vec{w}$ to $\vec{x}$ in the binding graph G_α. We say that π is a *pumpable path from $\vec{w}$ to*

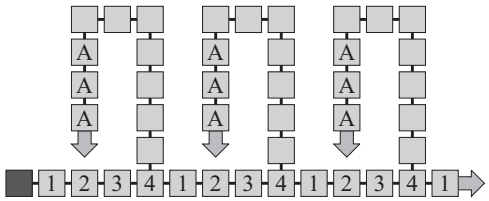


Figure 6.1: An assembly containing a path with repeating tiles A-A that do *not* form a pumpable segment, because they are blocked from infinite growth by the existing assembly. Note, however, that any sufficiently long path from the origin (at the left) contains a pumpable segment, namely the repeating segment 1-2-3-4-1 along the bottom row, which can be pumped infinitely to the right.

$\vec{x}$ *in* α if it contains a pumpable segment $\pi[i \dots k]$ for some $i, k \in \mathbb{N}$ such that $0 \leq i < k \leq |\pi|$. We say that $\mathcal{T}$ is *c-pumpable* if there exists $c \in \mathbb{N}$ such that for every $\vec{w}, \vec{x} \in \text{dom } \alpha$ with $\vec{x} \notin D(c, \vec{w})$, there exists a pumpable path π from $\vec{x}$ to $\vec{w}$ in α.

Observation 6.2.1. *If a directed TAS* $\mathcal{T} = (T, \sigma, 1)$ *has the property that no adjacent tiles in the terminal assembly have mismatched glues, then* $\mathcal{T}$ *is pumpable.*

Proof. Let α be the unique terminal assembly of $\mathcal{T}$. Let π be any path in G_α of length at least $|T| + 1$, and let $\vec{u}, \vec{v}$ be two positions of π with the same tile type. If $\mathcal{T}$ is not pumpable, then attempting to repeat π will cause a "collision" with some other position in α; i.e., eventually the repetition of π will result in attempting to place a tile type t at a position $\vec{x}$ at which $\alpha(\vec{x}) \neq t$. Let p be the tile type at the position immediately preceding $\vec{x}$ on the copy of π at which this occurs. Then the glue of p on the side facing $\vec{x}$ must mismatch with the opposite glue of $\alpha(\vec{x})$, or else in any *previous* repetition of π, t and $\alpha(\vec{x})$ would nondeterministically compete to bind to p, violating the directedness of $\mathcal{T}$. Since we assumed $\mathcal{T}$ is directed and has no mismatches, this implies $\mathcal{T}$ must be c-pumpable with $c = |T| + 1$. □

Figure 6.2 shows, from left to right, (1) a partially complete assembly beginning from the (dark grey) seed tile, where the dark notches between adjacent tiles represent strength 1 bonds, and a tile selected for the example, (2) the full path leading from the seed to the selected tile, (3) the tile types for a segment of the path, showing the repeating pattern of tile types '1-2-3-4', and (4) an extended version of the path which shows its ability to be pumped.

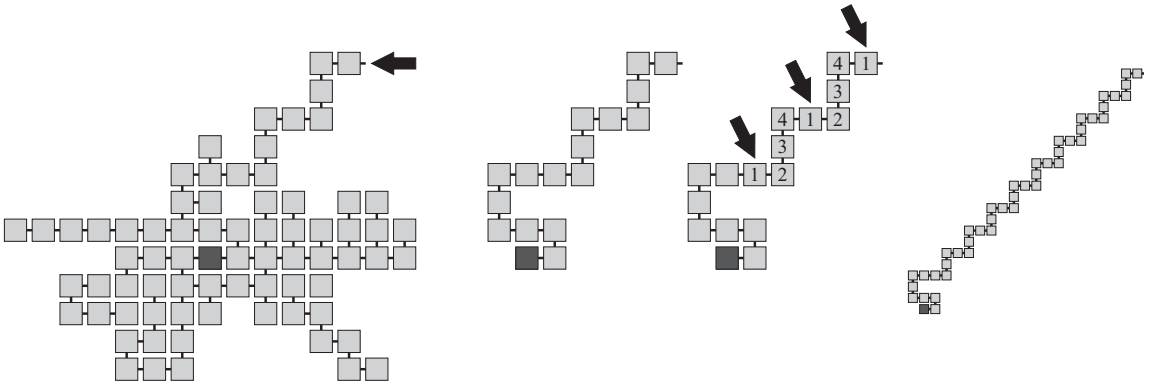


Figure 6.2: A partial assembly, selected path, pumpable segment, and pumpable path

In our proof, it is helpful to consider extending an assembly in such a way that no individual tile in the existing assembly is extended by more than a finite amount (though an infinite assembly may have an infinite number of tiles that can each be extended by a finite amount). We call such an extension the *finite closure* of the assembly, and define it formally as follows. Let $\alpha' \in \mathcal{A}[\mathcal{T}]$. We say that the *finite closure* of α' is the unique assembly $\mathcal{F}(\alpha')$ satisfying

1. $\alpha' \sqsubseteq \mathcal{F}(\alpha')$, and
2. dom $\mathcal{F}(\alpha')$ is the set of all points $\vec{x} \in \mathbb{Z}^2$ such that every infinite simple path in the binding graph G_α containing $\vec{x}$ intersects dom α'.

Intuitively, this means that if we extend α' by only those "portions" that will eventually stop growing, the finite closure is the super-assembly that will be produced. That is, any attempt to "leave" α' through the finite closure and go infinitely far will eventually run into α' again. If α' is terminal, then α' is its own finite closure. Note that in general, the finite closure of an assembly α' is not the result of adding finitely many tiles to α'. For instance, if infinitely many points of α' allow exactly one tile to be added, the finite closure adds infinitely many points to α'. However, the finite closure of a finite assembly is always a finite assembly.

For an example of a finite closure of an assembly, see Figure 6.3. Figure 6.3c shows the terminal assembly which consists of three rows of tiles that continue infinitely to the right (denoted by the arrow), with a 10 tile upward projection occurring at every fourth column.

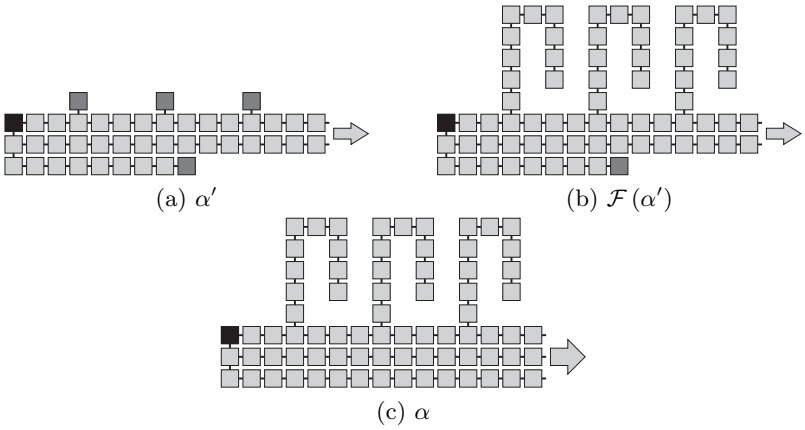


Figure 6.3: Example of a finite closure. The dark gray points represent locations at which tiles can attach.

Figure 6.3a shows an assembly α' which consists of two rows of tiles continuing infinitely to the right but with an incomplete bottom row and none of the full upward projections. Figure 6.3b shows $\mathcal{F}(\alpha')$, which is the finite closure of α'. Notice that an infinite number of tiles were added to α' to create $\mathcal{F}(\alpha')$ since an infinite number of upward projections were added, each consisting of only 10 tiles. However, also note that the bottom row is not grown because that row consists of an infinite path and therefore cannot be part of the finite closure.

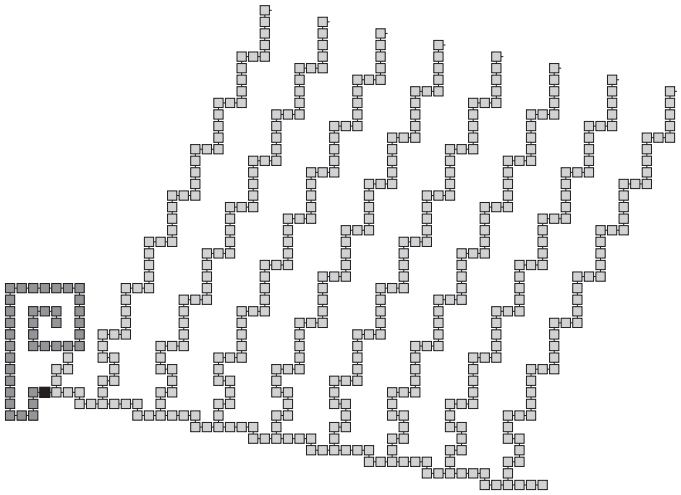


Figure 6.4: Example of a comb connected to a hard-coded assembly (dark tiles that spiral). The seed is the tile at the center of the spiral. The comb originates at the bottom most black tile. Note that a finite number of "attempts at teeth" (one in the above example) may be blocked before the infinite teeth are allowed to grow.

Let $\pi^{\rightarrow}$ be an eventually $\vec{v}$-semi-periodic path in α originating at $\vec{0}$ where $\vec{0} \neq \vec{v} \in \mathbb{Z}^2$. Let $\vec{u} \in \mathbb{Z}^2$ such that $\vec{u} \neq z \cdot \vec{v}$ for all $z \in \mathbb{R}$. Suppose that there is a point $\vec{b}$ on the tail of $\pi^{\rightarrow}$ such that there is an eventually $\vec{u}$-semi-periodic path $\pi^{\uparrow}$ in α originating at $\vec{b}$ such that $\pi^{\rightarrow} \cap \pi^{\uparrow} = \left\{\vec{b}\right\}$. Define the assembly $\alpha^* = \alpha \upharpoonright \left(\pi^{\rightarrow} \cup \bigcup_{n \in \mathbb{N}} \left(\pi^{\uparrow} + n \cdot \vec{v}\right)\right)$. It is easy to see that α^* need not be a producible assembly. We say that α^* is a *comb in* (with respect to $\pi^{\rightarrow}$ and $\pi^{\uparrow}$) if, for every $n \in \mathbb{N}$, $\alpha^* \upharpoonright \pi^{\uparrow} + n \cdot \vec{v} = \alpha^* \upharpoonright \left(\pi^{\uparrow} + n \cdot \vec{v}\right)$. We refer to the assembly $\alpha^* \upharpoonright \pi$ as the *base* of α^*. For any $n \in \mathbb{N}$, we define the n^{th} *tooth of* α^* to be the assembly $\alpha \upharpoonright \left(\pi^{\uparrow} + n \cdot \vec{v}\right)$. We say that the comb α^* *starts at the point* $\pi[s]$ (the point at which the eventually-periodic path $\pi^{\rightarrow}$ becomes periodic). It follows from the definition that, if α^* is a comb in α, then $\alpha^* \in \mathcal{A}[\mathcal{T}]$.

See Figure 6.4 for an example of a comb. A comb, intuitively, is a generalization of the assembly in which an infinite periodic one-way path (the base) grows along the positive x-axis, and once per period, an infinite periodic path (a tooth) grows in the positive y direction, creating an infinite number of "teeth". The generalizations are that (1) the base and teeth need not run parallel to either axis, and (2), the teeth may have some initial hard-coded tiles before the repeating periodic segment begins. Also, it is possible at temperature 1 to build multiple combs with the same base, but with different teeth, growing in either direction.

6.3 Main Result

We show in this section that only "simple" sets weakly self-assemble in directed, pumpable tile assembly systems at temperature 1. We now formally define "simple."

Definition 6.3.1. A set $X \subseteq \mathbb{Z}^2$ is *linear* if there exist three vectors $\vec{b}$, $\vec{u}$, and $\vec{v}$ such that

$$X = \left\{ \vec{b} + n \cdot \vec{u} + m \cdot \vec{v} \;\middle|\; n, m \in \mathbb{N} \right\}.$$

A set is *semilinear* if it is a finite union of linear sets.

Note that a set that is periodic along only one dimension is also linear, since this corresponds to the condition that exactly one of the vectors $\vec{u}$ or $\vec{v}$ is equal to $\vec{0}$ (or if $\vec{u} = \vec{v}$). Similarly, if

$\vec{u} = \vec{v} = \vec{0}$, then the definition of linear is equivalent to A being a singleton set. The following observation (as well as, for example, [17]) justifies the intuition that semilinear sets constitute only the computationally simplest subsets of $\mathbb{Z}^2$.

Observation 6.3.2. *Let $A \subseteq \mathbb{Z}^2$ be a semilinear set. Then the unary languages*

$$L_{A,x} = \left\{ \; 0^{|x|} \;\middle|\; (x, y) \in A \text{ for some } y \in \mathbb{Z} \; \right\}$$

and

$$L_{A,y} = \left\{ \; 0^{|y|} \;\middle|\; (x, y) \in A \text{ for some } x \in \mathbb{Z} \; \right\}$$

consisting of the unary representations of the projections of A onto the x-axis and y-axis, respectively, are regular languages.

A proof of Observation 6.3.2 is included in the technical appendix. The following theorem is the main result of this chapter.

Theorem 6.3.3. *Let $\mathcal{T} = (T, \sigma, 1)$ be a directed, pumpable TAS. If a set $X \subseteq \mathbb{Z}^2$ weakly self-assembles in $\mathcal{T}$, then X is a semilinear set. Conversely, every semilinear set weakly self-assembles at temperature 1.*

A proof of the forward direction of Theorem 6.3.3 is given in the technical appendix. The proof idea of Theorem 6.3.3 is as follows. Suppose that $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$. Note that α is unique since $\mathcal{T}$ is directed. Either α is an infinite "grid" that fills the plane, or there exists finitely many linear combs and paths that, taken together, "cover" every point in X (in the sense that each such point is in the finite closure of one of these combs or paths).

The reason for this is that each comb is defined by two vectors $\vec{u}$ (the base) and $\vec{v}$ (the teeth), and these vectors form a "basis" for the space of points located within the cone formed by the base and the first tooth of the comb. While the vectors do not reach every point in this cone, they reach within a constant distance of every point in the cone, and the doubly periodic regularity of the teeth and base enforces doubly periodic regularity in between the teeth as well. Of course, not all combs have teeth, in which case the comb is just a periodic path.

We associate each black tile with some periodic path or comb that begins in a fixed radius about the origin (utilizing the fact that a path cannot go far from the origin without having pumpable segments that can be used to construct a periodic path or comb). The finite number of combs and periodic paths originating within this radius tells us that the number of semilinear sets of (locations tiled by) black tiles that they each define is finite.

6.4 An Application to Discrete Self-Similar Fractals

In this section, we use Theorem 6.3.3 to show that no discrete self-similar fractal weakly self-assembles in any temperature 1 tile assembly system that is pumpable and directed. Since Winfree [45] showed that one particular discrete self-similar fractal, the discrete Sierpinkski triangle, self-assembles at temperature 2, this provides a concrete example of computation that is possible (and simple) at temperature 2, but impossible at temperature 1, assuming directedness and pumpability.

Definition 6.4.1. Let $1 < c \in \mathbb{N}$, and $X \subsetneq \mathbb{N}^2$ (we do not consider $\mathbb{N}^2$ to be a self-similar fractal). We say that X is a *c-discrete self-similar fractal*, if there is a set $V \subseteq \{0, \ldots, c-1\} \times \{0, \ldots, c-1\}$ with

$$V \notin \{\{(i,i) \mid 0 \leq i < c\}, \{(i,0) \mid 0 \leq i < c\}, \{(0,i) \mid 0 \leq i < c\}\}$$

such that

$$X = \bigcup_{i=0}^{\infty} X_i,$$

where X_i is the i^{th} *stage* satisfying $X_0 = \{(0,0)\}$, and $X_{i+1} = X_i \cup \left(X_i + c^i V\right)$. In this case, we say that V *generates* X.

Definition 6.4.2. $X \subsetneq \mathbb{N}^2$. We say that X is a *discrete self-similar fractal* if it is a c-discrete self-similar fractal for some $c \in \mathbb{N}$.

The following observation is clear by Definition 6.4.1.

Observation 6.4.3. *No discrete self-similar fractal is a semilinear set.*

Theorem 6.4.4. *Let $X \subsetneq \mathbb{N}^2$ be a discrete self-similar fractal.*

1. *X does not weakly self-assemble in any tile assembly system that is pumpable and directed.*
2. *X does not strictly self-assemble in any tile assembly system that is pumpable and directed.*

Proof. (1) follows directly from Observation 6.4.3. To prove (2), note that if a set $X \subseteq \mathbb{Z}^2$ weakly self-assembles then, by definition, X strictly self-assembles. □

6.5 Conclusion

We have studied the class of shapes that self-assemble in Winfree's abstract tile assembly model at temperature 1. We introduced the notion of a pumpable temperature 1 tile assembly system and then proved that, if X weakly self-assembles in a pumpable, directed tile assembly system, then X is necessarily "simple" in the sense that X is merely a semilinear set, a finite union of simple, "two-dimensionally periodic" sets. We used this result to prove that no discrete self-similar fractal weakly/strictly self-assembles in any pumpable, directed tile assembly system. Finally, we conjecture that our results hold in the absence of the pumpability hypothesis.

Conjecture 6.5.1. *Let $\mathcal{T} = (T, \sigma, 1)$ be a directed tile assembly system and $\alpha \in \mathcal{A}_\square[\mathcal{T}]$. If* dom α *is infinite, then $\mathcal{T}$ is pumpable.*

It is always possible to produce long paths in which the presence of a segment with two repetitions of a tile type does not imply that the segment is pumpable. However, in every case we consider, there is always a previous segment of the path that is pumpable. Proving Conjecture 6.5.1 would imply that every directed tile set weakly self-assembles a semilinear set.

We also leave open the question of whether the hypothesis of directedness may be removed. We use the property of directedness at many points in our proof, but in some cases, a more careful and technically convoluted argument could be used to show that the tile set need not

be directed. Intuitively, an undirected tile set $\mathcal{T}$ that weakly self-assembles a set $X \subseteq \mathbb{Z}^2$ is deterministic in that all terminal assemblies of $\mathcal{T}$ "paint" exactly the points in X black, but is nondeterministic in the sense that different terminal assemblies of $\mathcal{T}$ may place different tiles in the same location (including different black tiles at locations in X), and may even place non-black tiles at locations in one terminal assembly that are left empty in other terminal assemblies. Undirected tile assembly systems that weakly self-assemble a unique set X exist, but in every case that we know of, the undirected tile set may be replaced by a directed tile set self-assembling the same set.

If the hypothesis of directedness and pumpability could be removed from the entire proof, then our main result would settle the case of computation via self-assembly at temperature 1, by showing that every temperature 1 tile assembly system weakly self-assembles a semilinear set if it weakly self-assembles any set at all. As indicated in the introduction, we interpret this statement to imply that general-purpose computation is not possible with temperature 1 tile assembly systems.

Finally, we make the observation that universal computation at temperature 1 *is* possible by changing the underlying model of self-assembly. There is a different sense in which undirected temperature 1 tile assembly systems may be considered to achieve "universal computation". Adleman has devised a nondeterministic temperature 1 tile assembly system, cited as a personal communication in Rothemund's Ph.D. thesis [31], which does something similar to the following (the exact construction described next is reconstructed from conversations and may differ in details from Adleman's original construction). It simulates a single-tape Turing machine in the standard way, by assembling a space-time diagram of the configuration history of the Turing machine, with the growth front of the assembly crawling back and forth across the representation of the tape at time t (the t^{th} row of the assembly), in order to construct the tape at time $t + 1$. Each cell encodes a value consisting of a tape symbol, the boolean value "Is the tape head here?", and if the tape head is present, what the state is. In order to read the value of a cell, the tile assembly system nondeterministically guesses the value. Any nondeterministic choice failing to correctly predict the value terminates the growth of

the assembly, while correctly guessing continues the growth. Hence, many assemblies will be produced, but there is a unique largest assembly (infinite if the computation of the Turing machine is infinite), which represents the result of the computation. However, this construction does not weakly self-assemble any set representing the result of the computation, as the definition of weak self-assembly, while not requiring the tile assembly system to be directed, does require a unique set to be weakly assembled. So the question of whether nondeterministic computation can achieve universal computation has been answered, but in a weaker form than via the mechanism of weak self-assembly.

In addition, a construction that was personally communicated by Matthew Cook (also mentioned briefly in [31]), which is based on Adleman's nondeterministic construction, establishes that the aTAM is computationally universal with respect to directed, temperature 1 tile assembly systems that place tiles in three spatial dimensions. In fact, Cook's construction only uses two integer planes, "stacked" one on top of the other, to simulate an arbitrary Turing machine (no such directed two-dimensional universality result is known at the time of this writing).

Moreover, we show that universality can also be achieved if negative glue strengths and interaction between differently colored glues (a so-called *non-diagonal strength function*) are allowed.

Theorem 6.5.2. *For every single-tape Turing machine M, there is a tile set T with negative, non-diagonal glue strengths, which simulates M in the following way. Given an input string x, define $\mathcal{T}_x = (T, \sigma_x, 1)$ to be the temperature 1 TAS where σ_x is the seed assembly satisfying* $\text{dom } \sigma_x = \{0, \ldots, |x| - 1\} \times \{0\}$ *that encodes the initial configuration of M. Then $\mathcal{T}_x$ simulates the computation of $M(x)$, with the configuration of $M(x)$ after n steps represented by the line $y = n$ in the terminal assembly of $\mathcal{T}_x$.*

A proof sketch of Theorem 6.5.2 is given in the technical appendix. Intuitively, by introducing negative glue strengths, we allow for the cooperation that is impossible with only nonnegative glue strengths. The difference with temperature 2 is that, at temperature 2, one may enforce that no tile appears at a certain position until two neighbors are present. At

temperature 1 and with negative glue strengths, on the contrary, we do not enforce that two tiles are present before a position can be given a tile. However, we enforce that if the wrong tile binds in this position, eventually the error is corrected by the presence of a neighboring tile which forces the removal of the incorrect tile using negative glue strengths.

6.6 Technical Appendix

This appendix contains proofs of results that were not given in the main text, and some definitions required only for those proofs.

Proof. (*of Observation 6.3.2*) Let $B \subseteq \mathbb{Z}^2$ be a linear set. It is routine to verify that B's projection along the x-axis is a (singly) periodic set; i.e., a set

$$B_x = \{\ x \in \mathbb{Z} \mid (x, y) \in B \text{ for some } y \in \mathbb{Z}\ \}$$

such that there is a number $v \in \mathbb{N}$ such that, for all $x \in \mathbb{Z}$, $x \in B_x \implies x + v \in B_x$. It is well-known that a unary language $L \in \{0\}^*$ is regular if and only if the set $N = \{\ n \in \mathbb{N} \mid 0^n \in L\ \}$ of lengths of strings in L is eventually periodic. Therefore the language $L_{B,x} = \left\{\ 0^{|n|} \;\middle|\; n \in B_x\ \right\}$ is regular. A symmetric argument establishes that $L_{B,y}$ is a regular language as well, so the theorem holds for any linear set. Since A is a finite union of linear sets, the theorem follows by the closure of the regular languages under finite union. □

We will now prove a series of technical lemmas that will reveal "order" in the seemingly disordered realm of temperature 1 (a.k.a., non-cooperative) self-assembly.

Definition 6.6.1. Let α_1, α_2 be assemblies. We say that α_1 and α_2 are *consistent* if, for all $\vec{v} \in \text{dom } \alpha_1 \cap \text{dom } \alpha_2$, $\alpha_1(\vec{v}) = \alpha_2(\vec{v})$.

Definition 6.6.2. Let α_1, α_2 be consistent assemblies. The *union of* α_1 *and* α_2, written as $\alpha_1 \cup \alpha_2$, is the unique assembly $\alpha_1 \cup \alpha_2$ satisfying, dom $(\alpha_1 \cup \alpha_2) =$ dom $\alpha_1 \cup$ dom α_2 and $\alpha_1 \sqsubseteq \alpha_1 \cup \alpha_2$ and $\alpha_2 \sqsubseteq \alpha_1 \cup \alpha_2$.

Observation 6.6.3. *Let $\mathcal{T} = (T, \sigma, 1)$ be a directed TAS. If α_1 and α_2 are consistent, connected assemblies, and $\alpha_1 \in \mathcal{A}[\mathcal{T}]$, then $\alpha_1 \cup \alpha_2 \in \mathcal{A}[\mathcal{T}]$.*

The next lemma states that, if we wish to show that the existence of a semi-periodic path in the assembly implies a periodic path (i.e., it is not only one-way infinite but two-way infinite, bisecting the plane), it suffices to show that a translation – along the path – of the assembly connecting the seed to the path, are consistent with the path, so long as the translation is sufficiently far to achieve maximal intersection between the assembly and the path.

Lemma 6.6.4. *Let $\mathcal{T} = (T, \sigma, 1)$ be a directed TAS, $\vec{0} \neq \vec{v} \in \mathbb{Z}^2$, α be the unique assembly satisfying $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$, and $\vec{a} \in \text{dom } \alpha$. Let $\pi_{\vec{a}}^{\rightarrow}$ be a $\vec{v}$-periodic path in α originating at $\vec{a}$, and let $\pi_{\vec{0},\vec{a}}$ be a simple finite path in G_α from $\vec{0}$ to $\vec{a}$. Let $c > |\pi_{\vec{0},\vec{a}}|$ be a positive integer. If $\left(\alpha \upharpoonright \pi_{\vec{0},\vec{a}}\right) + c \cdot \vec{v}$ is consistent with $\alpha \upharpoonright \pi_{\vec{a}}^{\rightarrow}$, then there is a (two-way infinite) $\vec{v}$-periodic path in G_α containing $\vec{a}$.*

Proof. Since $c > \left|\pi_{\vec{0},\vec{a}}\right|$, dom $\left(\left(\alpha \upharpoonright \pi_{\vec{0},\vec{a}}\right) + c \cdot \vec{v}\right) \cap \pi_{\vec{a}}^{\rightarrow}$ is maximal over all $c \in \mathbb{N}$. Since $\left(\alpha \upharpoonright \pi_{\vec{0},\vec{a}}\right) + c \cdot \vec{v}$ is consistent with $\alpha \upharpoonright \pi_{\vec{a}}^{\rightarrow}$, extending the tiles of $\pi_{\vec{a}}^{\rightarrow}$ in the direction $-\vec{v}$ will intersect $\pi_{\vec{0},\vec{a}}$ only at each position where the existing tile in $\alpha \upharpoonright \pi_{\vec{0},\vec{a}}$ agrees with the extension. Therefore such an extension will not be blocked by $\alpha \upharpoonright \pi_{\vec{0},\vec{a}}$, and $\pi_{\vec{a}}^{\rightarrow}$ is merely one half of a (two-way) periodic path in α. □

The next lemma states that if two parallel semi-periodic paths are connected through their pumpable segments in "another way besides the trivial way" (since all points are connected by *some* path in the binding graph), then this connection can be exploited to show that the paths must actually form two-way periodic paths that bisect the plane.

Lemma 6.6.5. *Let $\mathcal{T} = (T, \sigma, 1)$ be a directed TAS, $\vec{0} \neq \vec{v} \in \mathbb{Z}^2$, α be the unique assembly satisfying $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$, $\vec{a} \in \text{dom } \alpha$ and $\pi_{\vec{0},\vec{a}}$ be a simple finite path from $\vec{0}$ to $\vec{a}$ in α. If $\pi_{\vec{a}}^{\rightarrow}$ is a $\vec{v}$-semi-periodic path in G_α originating at $\vec{a}$, $\vec{d} \in \text{dom } \alpha$ such that $\pi_{\vec{d}}^{\rightarrow}$ is a $z \cdot \vec{v}$-semi-periodic path in α originating at $\vec{d}$, for some $z \in \mathbb{R}^+$, and there is a simple path from some point $\vec{c}$ on the tail of $\pi_{\vec{a}}^{\rightarrow}$ to $\vec{d}$ in G_α, denoted as $\pi_{\vec{c},\vec{d}}$, with $\pi_{\vec{c},\vec{d}} \cap \left(\pi_{\vec{0},\vec{a}} \cup \pi_{\vec{a}}^{\rightarrow} \cup \pi_{\vec{d}}^{\rightarrow}\right) = \left\{\vec{c}, \vec{d}\right\}$, then there exists a $\vec{v}$-periodic path in α containing $\vec{a}$.*

Intuitively, we prove Lemma 6.6.5 by showing that an appropriately translated copy of the simple (finite) path from the seed to the origination point of $\pi^{\rightarrow}_{\vec{a}}$ is consistent with $\pi^{\rightarrow}_{\vec{a}}$. See Figure 6.5 for a visual depiction of (a simple example of) the hypothesis of Lemma 6.6.5 and Figure 6.6 for an illustration of the conclusion of Lemma 6.6.5.

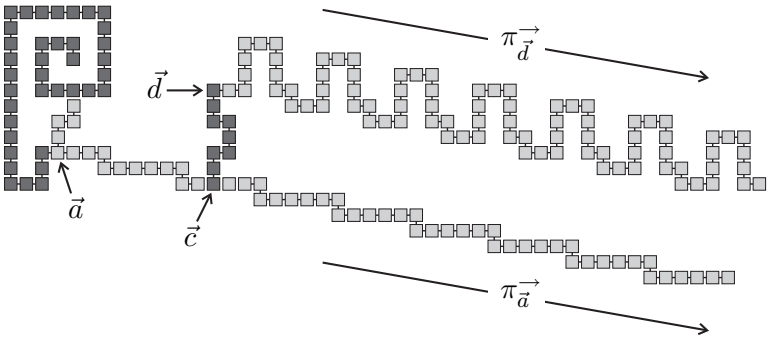


Figure 6.5: An example of the hypothesis of Lemma 6.6.5. The seed tile is at the center of the "spiral." The path $\pi_{\vec{c},\vec{d}}$ is represented by the dark tiles that form the "bridge" between the tail of $\pi^{\rightarrow}_{\vec{a}}$ (the lowest $\vec{v}$-periodic path) and some point on $\pi^{\rightarrow}_{\vec{d}}$ (the upper $\vec{v}$-periodic path).

Proof. Let $c \in \mathbb{N}$ such that $c > \left|\pi_{\vec{0},\vec{a}}\right|$. By Lemma 6.6.4, it suffices to show that $\left(\alpha \upharpoonright \pi_{\vec{0},\vec{a}}\right) + c \cdot \vec{v}$ is consistent with $\alpha \upharpoonright \pi^{\rightarrow}_{\vec{a}}$. If consistency holds, then we are done, so assume otherwise. This means that there is a point

$$\vec{b} \in \left(\pi_{\vec{0},\vec{a}} + c \cdot \vec{v}\right) \cap \pi^{\rightarrow}_{\vec{a}},$$

satisfying the following two conditions.

1. $\left(\left(\alpha \upharpoonright \pi_{\vec{0},\vec{a}}\right) + c \cdot \vec{v}\right)\left(\vec{b}\right) \neq \left(\alpha \upharpoonright \pi^{\rightarrow}_{\vec{a}}\right)\left(\vec{b}\right)$, and
2. $\vec{b}$ is the "closest" point to $\vec{a} + c \cdot \vec{v}$ on the path $\pi_{\vec{0},\vec{a}} + c \cdot \vec{v}$.

Moreover, the assumption that $\pi_{\vec{c},\vec{d}} \cap \left(\pi_{\vec{0},\vec{a}} \cup \pi^{\rightarrow}_{\vec{a}} \cup \pi'\right) = \left\{\vec{c}, \vec{d}\right\}$, along with the existence of the $\vec{v}$-periodic path $\pi^{\rightarrow}_{\vec{d}}$, can be used to show that there exists a simple cycle C in G_α that contains the point $\vec{b}$ such that not every simple (finite) path from $\vec{0}$ to (any point in) C goes through $\vec{b}$. This means that it is possible to define an assembly sequence where (the tile placed at) the input neighbor of $\vec{b}$ is the same as the (tile placed at) the output neighbor of $\vec{b} - c \cdot \vec{v}$ in

the assembly sequence resulting in $\alpha \upharpoonright \pi_{\vec{0},\vec{a}}$. Since $\mathcal{T}$ is directed, these must be the same tiles, whence there can be no "first" point of inconsistency between $\left(\alpha \upharpoonright \pi_{\vec{0},\vec{a}}\right) + c \cdot \vec{v}$ and $\alpha \upharpoonright \pi_{\vec{a}}^{\rightarrow}$. $\square$

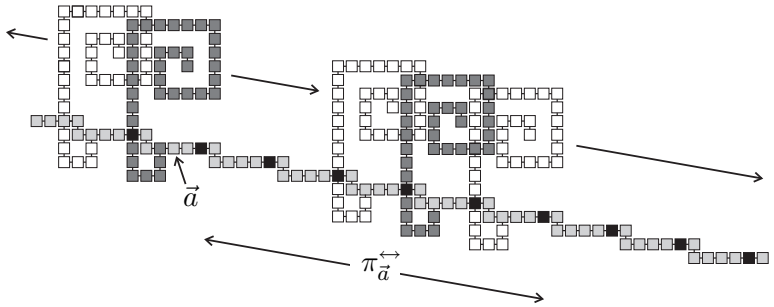


Figure 6.6: An example of the conclusion of Lemma 6.6.5. The path $\pi_{\vec{a}}^{\rightarrow}$ is a (two-way infinite) $\vec{v}$-periodic path, which means that (copies of) the path $\pi_{\vec{0},\vec{a}}$ can be translated in the $\pm\vec{v}$ direction.

Definition 6.6.6. Let α be an assembly. We say α is *doubly periodic* if there exist $\vec{u}, \vec{v} \in \mathbb{Z}^2$ such that $\vec{u} \neq \vec{v}$, $\vec{u} \neq \vec{0}$, $\vec{v} \neq \vec{0}$, and for all $\vec{a} \in \mathbb{Z}^2$, $\alpha(\vec{a}) = \alpha(\vec{a} + \vec{u}) = \alpha(\vec{a} + \vec{v})$, where $\alpha(\vec{a}) = \alpha(\vec{b})$ if both $\alpha(\vec{a})$ and $\alpha(\vec{b})$ are both undefined.

In other words, α is doubly periodic if its values form a repeating "grid" of parallelogram patterns on $\mathbb{Z}^2$, infinite in all directions. It is easy to see that, if α is doubly periodic, then for any set of tile types $B \subseteq T$, dom $\alpha(B)$ (the set that weakly self-assembles) is a union of four linear sets of integer lattice points.

The following technical lemma shows that, if a tail of a tooth of a comb connects to the base through a path other than the "natural" one, then the entire assembly is doubly periodic. This allows us to assume, in the proof of our main theorem, that the teeth of a comb are not interconnected in "inconvenient" ways.

Lemma 6.6.7. *Let $\mathcal{T} = (T, \sigma, 1)$ be a directed TAS, $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$, and α^* be a comb in α. If the tail of the n^{th} tooth is connected to (any point in) the $(n+1)^{st}$ tooth (for $n > 1$) via a simple finite path that does not go through any point on the n^{th} tooth, then α is doubly periodic.*

The proof idea of Lemma 6.6.7 is as follows. Suppose that α is the unique terminal assembly satisfying $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$. Since the tails of two different teeth of the comb α^* are connected via

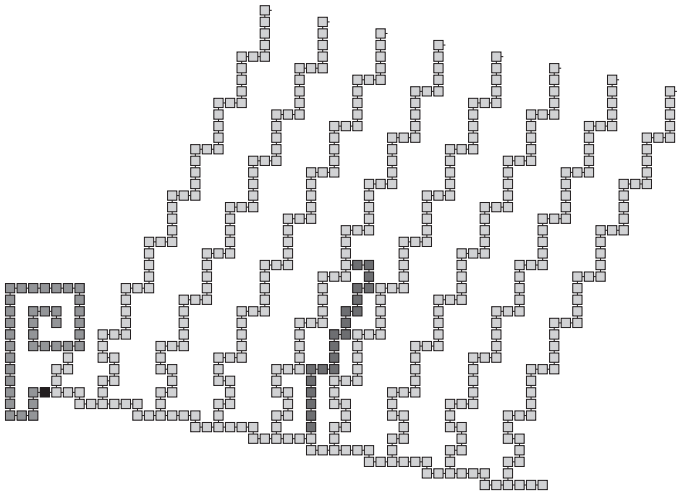


Figure 6.7: An example of a comb having two teeth connected via a simple path that does not go through the finite closure of the base. The seed tile is the tile in the center of the "spiral".

a simple path, we can build an assembly sequence whose result, denoted as $\alpha_{\#}$, is an infinite, plane-filling, doubly periodic assembly. We then show that this assembly is consistent with the tiles near the seed, which means it must be a producible assembly. Finally, since $\mathcal{T}$ is directed and $\alpha_{\#} \sqsubseteq \alpha$, we conclude that α is doubly periodic. Figure 6.7 illustrates (a simple example of) the hypothesis of the lemma.

Proof. Let α^* be a comb in α with respect to $\pi^{\rightarrow}$ and $\pi^{\uparrow}$, and $\vec{a}$ be the starting point of α^*. Assume that, for non-zero vectors $\vec{u} \neq \vec{v}$, $\pi^{\rightarrow}$ is an eventually $\vec{u}$-semi-periodic path in α originating at $\vec{0}$, and $\pi^{\uparrow}$ is an eventually $\vec{v}$-semi-periodic path in α originating at $\vec{b} \in \pi^{\rightarrow}$ with $\pi^{\rightarrow} \cap \pi^{\uparrow} = \left\{\vec{b}\right\}$.

The hypothesis says that, for some $1 < n \in \mathbb{N}$, there is a simple finite path from some point $\vec{c}$ on the tail of $\pi^{\uparrow} + n \cdot \vec{u}$ to some point $\vec{d} \in \pi^{\uparrow} + (n+1) \cdot \vec{u}$ that does not go through any point in $\pi^{\uparrow} + n \cdot \vec{u}$ except for the first point on $\pi_{\vec{c},\vec{d}}$. Denote this path as $\pi_{\vec{c},\vec{d}}$. Lemma 6.6.5 tells us that there exists a (two-way infinite) $\vec{v}$-periodic path in α containing the point $\widehat{\vec{b}} = \vec{b} + n \cdot \vec{u}$. Denote this path as $\pi^{\updownarrow}_{\widehat{\vec{b}}}$.

Let $\pi^{\rightarrow}_{\widehat{\vec{b}}}$ be the $\vec{u}$-semi-periodic path in G_α originating at $\widehat{\vec{b}}$. Since $\pi^{\updownarrow}_{\widehat{\vec{b}}}$ is $\vec{v}$-periodic and $\pi^{\updownarrow}_{\widehat{\vec{b}}} \cap \pi^{\rightarrow} \neq \varnothing$, it must be the case that there is a $\vec{u}$-semi-periodic path, denoted as $\pi^{\rightarrow}_{\widehat{\vec{b}}+\vec{v}}$, in α

originating at the point $\widehat{\vec{b}}+\vec{v}$. Let $\pi_{\vec{0},\widehat{\vec{b}}}$ be a finite simple path from $\vec{0}$ to $\widehat{\vec{b}}$ in G_α. If we (re)define $\vec{c} = \widehat{\vec{b}}$, $\vec{d} = \widehat{\vec{b}}+\vec{v}$, and note that there is a simple (finite) path from $\vec{c}$ on the tail of $\pi^{\rightarrow}$ (because $n > 1$) to $\vec{d} \in \pi^{\rightarrow}_{\widehat{\vec{b}}+\vec{v}}$, denoted as $\pi_{\vec{c},\vec{d}}$, in G_α with $\pi_{\vec{c},\vec{d}} \cap \left(\pi_{\vec{0},\widehat{\vec{b}}} \cup \pi^{\rightarrow}_{\widehat{\vec{b}}} \cup \pi^{\rightarrow}_{\widehat{\vec{b}}+\vec{v}}\right) = \left\{\vec{c},\vec{d}\right\}$, then we can again use Lemma 6.6.5 to conclude that there is a (two-way infinite) $\vec{v}$-periodic path containing $\widehat{\vec{b}}$, hence also containing $\vec{a}$ since those points differ by a multiple of $\vec{v}$. Denote this path as $\pi^{\leftrightarrow}_{\vec{a}}$. If $\pi_{\vec{0},\vec{a}}$ is the initial segment of $\pi^{\rightarrow}$, then the assembly $\alpha \upharpoonright \left(\pi_{\vec{0},\vec{a}} \cup \pi^{\leftrightarrow}_{\vec{a}} \cup \pi^{\updownarrow}_{\vec{b}}\right) \in \mathcal{A}[\mathcal{T}]$.

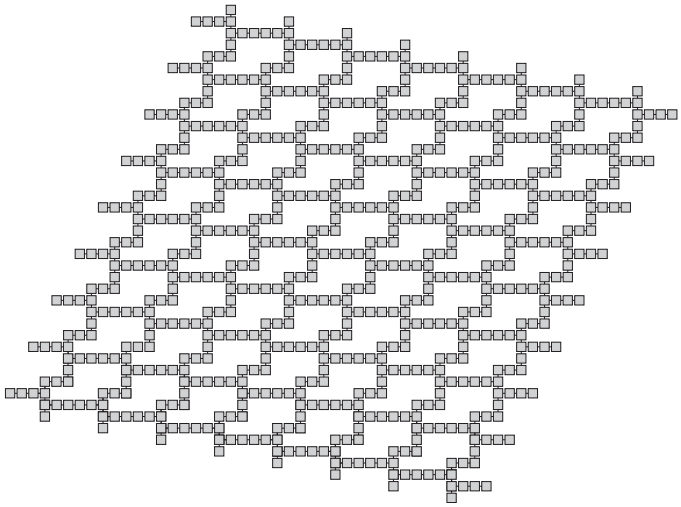


Figure 6.8: A finite sub-assembly of $\alpha_{\#}$.

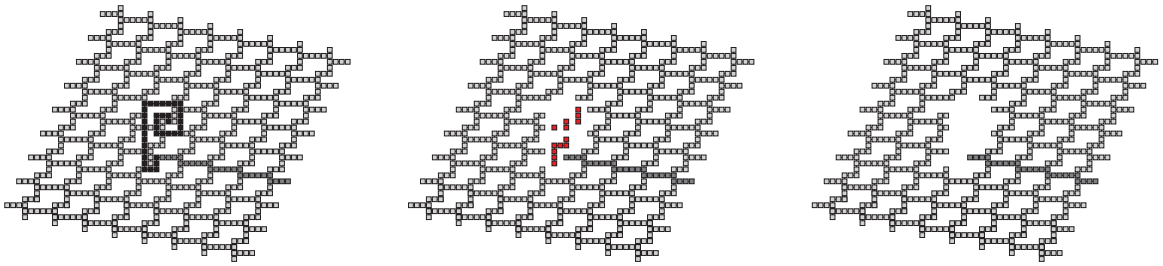
Define the assembly $\alpha_{\#}$ as follows.

$$\text{dom } \alpha_{\#} = \bigcup_{n\in\mathbb{Z}} ((\alpha \upharpoonright \pi^{\leftrightarrow}_{\vec{a}}) + n\cdot\vec{v}) \cup \bigcup_{n\in\mathbb{Z}} \left(\left(\alpha \upharpoonright \pi^{\updownarrow}_{\vec{b}}\right) + n\cdot\vec{u}\right).$$

Our goal is to show that $\alpha_{\#} \cup \left(\alpha \upharpoonright \pi_{\vec{0},\vec{a}}\right) \in \mathcal{A}[\mathcal{T}]$, which will nearly complete the proof since $\alpha_{\#}$ is doubly periodic.

Let $\alpha_{\text{almost-}\#}$ be the largest assembly satisfying $\alpha_{\text{almost-}\#} \sqsubseteq \alpha_{\#}$ such that dom $\alpha_{\text{almost-}\#} \cap \pi_{\vec{0},\vec{a}} = \varnothing$ and $G_{\alpha_{\text{almost-}\#}}$ is connected. Note that the assembly $\alpha_{\text{almost-}\#} \cup \left(\alpha \upharpoonright \pi_{\vec{0},\vec{a}}\right) \in \mathcal{A}[\mathcal{T}]$. See Figure 6.9 for an example of how the assembly $\alpha_{\text{almost-}\#}$ is constructed.

In order to prove that $\alpha_{\#} \cup \left(\alpha \upharpoonright \pi_{\vec{0},\vec{a}}\right) \in \mathcal{A}[\mathcal{T}]$, it suffices to show that $\alpha \upharpoonright \pi_{\vec{0},\vec{a}}$ and $\alpha_{\#}$ are consistent, since $\alpha \upharpoonright \pi_{\vec{0},\vec{a}} \in \mathcal{A}[\mathcal{T}]$ and $\pi_{\vec{0},\vec{a}} \cap \text{dom } \alpha_{\#} \neq \varnothing$. Note that, by the way we constructed



(a) $\alpha_{\#} \cup (\alpha \upharpoonright \pi_{\vec{0},\vec{a}})$ (b) $\alpha_{\#} - (\alpha \upharpoonright \pi_{\vec{0},\vec{a}})$ (c) $\alpha_{\text{almost}-\#}$

Figure 6.9: The black tiles are the path $\pi_{\vec{0},\vec{a}}$. Since $\alpha_{\#} - \left(\alpha \upharpoonright \pi_{\vec{0},\vec{a}}\right)$ may have a finite number of disconnected "islands", we remove these as well to get the largest *connected* subassembly of $\alpha_{\#}$.

$\alpha_{\#}$, not *every* point in $\pi_{\vec{0},\vec{a}}$ can be a point of inconsistency between $\alpha_{\#}$ and $\alpha \upharpoonright \pi_{\vec{0},\vec{a}}$. Fix $n^* \in \mathbb{N}$ such that $\pi_{\vec{0},\vec{a}} \cap \left(\pi_{\vec{0},\vec{a}} + (n^* \cdot \vec{u} + n^* \cdot \vec{v})\right) = \varnothing$, which exists by the finiteness of $\pi_{\vec{0},\vec{a}}$. Since there exists at least one point $\vec{s} \in \pi_{\vec{0},\vec{a}} \cap \operatorname{dom} \alpha_{\#}$ such that $\alpha_{\#}(\vec{s}) = \left(\alpha \upharpoonright \pi_{\vec{0},\vec{a}}\right)(\vec{s})$, we can conclude that

$$\left(\alpha_{\text{almost-}\#} \cup \left(\alpha \upharpoonright \pi_{\vec{0},\vec{a}}\right)\right) \cup \left(\left(\alpha \upharpoonright \pi_{\vec{0},\vec{a}}\right) + n^* \cdot \vec{u} + n^* \cdot \vec{v}\right) \in \mathcal{A}[\mathcal{T}]$$

because $\mathcal{T}$ is directed and every point in $\pi_{\vec{0},\vec{a}} + n^* \cdot \vec{u} + n^* \cdot \vec{v} \cap \operatorname{dom} \alpha_{\text{almost-}\#}$ is reachable from $\vec{0}$ via at least two simple finite paths. Thus, $\alpha_{\#}$ and $\alpha \upharpoonright \pi_{\vec{0},\vec{a}}$ must be consistent. Since $\alpha_{\#}$ is trivially doubly periodic, it follows that $\alpha_{\#} \cup \left(\alpha \upharpoonright \pi_{\vec{0},\vec{a}}\right) \in \mathcal{A}[\mathcal{T}]$.

Since $\alpha_{\#} \sqsubseteq \alpha$, it suffices to show that the tiles added to $\alpha_{\#}$ to create α are doubly periodic. Since $\alpha_{\#}$ forms an infinite grid that partitions $\mathbb{Z}^2$ into infinitely many identical "parallelograms", bordered by the same tiles $\left(\text{those forming a single period of } \pi^{\leftrightarrow}_{\vec{a}} \text{ and } \pi^{\updownarrow}_{\vec{b}}, \text{ respectively}\right)$, then by the directedness of $\mathcal{T}$, each of these parallelograms must have the same tiles in the same positions relative to the borders of the parallelogram. Since the parallelogram borders are doubly periodic, the contents within the borders are doubly periodic as well, whence α is doubly periodic. □

The following lemma states that if there is a path from an assembly α' to a point not on the finite closure of α', then there is an eventually periodic path, originating from the same point as the first path, that intersects α' only at the first point.

Lemma 6.6.8. *Let $\mathcal{T} = (T, \sigma, 1)$ be a pumpable directed TAS, α be the unique assembly satisfying $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$, $\alpha' \sqsubseteq \alpha$, and $\vec{x} \in \text{dom } \alpha$. If $\vec{x} \notin \mathcal{F}(\alpha')$ and there is a finite simple path in G_α that goes through some point $\vec{s} \in \alpha'$ to $\vec{x}$, then, for some $\vec{0} \neq \vec{v} \in \mathbb{Z}^2$, there exists an eventually $\vec{v}$-semi-periodic path π in G_α, originating at $\vec{s}$, with $\pi \cap \text{dom } \alpha' = \{\vec{s}\}$.*

Proof. Let $\vec{r} \in \text{dom } \alpha'$ be a point connected to $\vec{x}$ by a simple finite path $\pi_{\vec{r},\vec{x}}$ satisfying $\pi_{\vec{r},\vec{x}} \cap \text{dom } \alpha' = \{\vec{r}\}$. Let $\vec{p}_0$ be the point on $\pi_{\vec{r},\vec{x}}$ closest to $\vec{x}$ such that G_α has an infinite simple path $\pi_{\vec{p}_0,\infty}$ starting at $\vec{p}_0$ that does not contain the point immediately before $\vec{p}_0$ on $\pi_{\vec{r},\vec{x}}$. Let $\vec{p}_1 \in \text{dom } \alpha \cap \left(\pi_{\vec{p}_0,\infty} - D(c, \vec{p}_0)\right)$, and $\pi_{\vec{r},\vec{p}_0,\vec{p}_1}$ be a simple path in G_α from $\vec{r}$ to $\vec{p}_1$ that goes through $\vec{p}_0$. Since $\mathcal{T}$ is pumpable, and $\left(\pi_{\vec{p}_0,\infty} - D(c, \vec{p}_0)\right)$, $\pi_{\vec{r},\vec{p}_0,\vec{p}_1}$ is a pumpable path. Let $\pi_{\vec{r},\vec{p}_0,\vec{p}_1}[i \ldots k]$, for $0 \leq i < k \leq \left|\pi_{\vec{r},\vec{p}_0,\vec{p}_1}\right|$, be the first pumpable segment of $\pi_{\vec{r},\vec{p}_0,\vec{p}_1}$, and $\vec{v} = \pi_{\vec{r},\vec{p}_0,\vec{p}_1}[k] - \pi_{\vec{r},\vec{p}_0,\vec{p}_1}[i]$. The lemma follows by letting $\vec{s} = \pi_{\vec{r},\vec{p}_0,\vec{p}_1}[i]$, and π be the $\vec{v}$-semi-periodic path originating at the point $\vec{s}$ defined by the pumpable segment. □

The next lemma states that if two teeth in a comb are connected in an "inconvenient" way (i.e., other than the trivial ways in which any two points in a stable assembly must be connected), then the entire assembly is an (two-way, two-dimensional) doubly periodic grid.

Lemma 6.6.9. *Let $\mathcal{T} = (T, \sigma, 1)$ be a pumpable directed TAS, α be the unique assembly satisfying $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$, and α^* be a comb in α. If there exists $\vec{x} \in \text{dom } \alpha$ such that there is a simple finite path from $\vec{0}$ to $\vec{x}$ that goes through the tail of some tooth of α^*, then $\vec{x} \in \mathcal{F}(\alpha^*)$ or α is doubly periodic.*

Proof. Let $\pi^{\uparrow}$ be the first tooth of α^*. Assume that $\vec{x} \notin \mathcal{F}(\alpha^*)$. It suffices to show that α is doubly periodic. Let $\vec{r}$ be an element of the tail of $n \cdot \vec{u} + \pi^{\uparrow}$ for some $n \in \mathbb{N}$, such that there exists a simple path in G_α from $\vec{r}$ to $\vec{x}$, denoted as $\pi_{\vec{r},\vec{x}}$, with $\pi_{\vec{r},\vec{x}} \cap \text{dom } \alpha^* = \{\vec{r}\}$. Such a point $\vec{r}$ exists because $\vec{x} \notin \mathcal{F}(\alpha^*)$. By Lemma 6.6.8, there exists an eventually $\vec{w}$-semi-periodic π in G_α, originating at $\vec{r}$ with $\pi \cap \text{dom } \alpha^* = \varnothing$.

If $\vec{w} \neq z \cdot \vec{v}$ for some $z \in \mathbb{R}^+$, then the $\vec{w}$-semi-periodic path it defines intersects a tooth of α^* and, by Lemma 6.6.7, α is doubly-periodic. Therefore assume that $\vec{w} = z \cdot \vec{v}$ for some $z \in \mathbb{R}^+$. Let $\vec{a} = \pi[s]$ - the point at which π becomes periodic. Lemma 6.6.5 tells us that there

exists a (two-way infinite) $\vec{w}$-periodic path in α containing the point $\vec{a}$. Denote this path as $\pi^{\updownarrow}_{\vec{a}}$. Let $\vec{b}$ be the closest point to $\vec{a}$ on the path $\pi^{\updownarrow}_{\vec{a}}$ that intersects the base of α^*. Let $\widehat{\pi}^{\uparrow}$ be the $\vec{w}$-semi-periodic path originating at $\vec{b}$ that does not intersect any teeth of α^*. Then we can form a new comb $\widehat{\alpha^*}$ with respect to $\pi^{\rightarrow}$ and $\widehat{\pi}^{\uparrow}$. Finally, we can apply Lemma 6.6.7 to either α^* or $\widehat{\alpha^*}$ in order to conclude that α is doubly periodic. □

The following observation is helpful in the proof of our main theorem, and states the obvious fact that combs define linear sets.

Observation 6.6.10. *Let $\mathcal{T} = (T, \sigma, 1)$ be a directed pumpable TAS in which the set $X \subseteq \mathbb{Z}^2$ weakly self-assembles, and let α be the unique terminal assembly satisfying $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$. If $\alpha^* \in \mathcal{A}[\mathcal{T}]$ is a comb in α, then the set $X \cap \operatorname{dom} \alpha^*$ is a semilinear set.*

Proof. (*of Theorem 6.3.3*) Assume the hypothesis. Let $c \in \mathbb{N}$ testify to the c-pumpability of $\mathcal{T}$. We denote by D_1, D_2, and D_3 the diamonds of radius c, $2c$, and $3c$, respectively, around the origin.

We show that each $\vec{x} \in X$ outside of D_2 is part of the finite closure of some periodic path or comb originating in D_3. Therefore, the union of the black tiles of the finite closure of each comb starting in D_3, each periodic path starting in D_3, and each singleton set containing a black tile in D_3, is the union proving the theorem. There are only a finite number of points in D_3, though each could be the origination point of multiple combs or periodic paths. However, if any such point has an infinite number of combs or periodic paths originating from it, then some will intersect at a different point, creating a cycle between the tails of two teeth of combs, and Lemma 6.6.7 tells us that X is doubly periodic. Otherwise, only a finite number of combs and periodic paths can originate in D_3, each one defining a term in the union, showing that the union is finite. Recall that a singleton set and a semi-periodic path are linear sets, and Observation 6.6.10 allows us to conclude that the remaining terms representing combs are linear as well.

Let $\vec{x} \in X - D_1$, and α be the unique assembly satisfying $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$. Since $\vec{x} \notin D_1$, and because $\mathcal{T}$ is c-pumpable, there exists a c-pumpable path from $\vec{0}$ to $\vec{x}$ in α. Denote this path

as $\pi_{\vec{0},\vec{x}}$. Let $\pi_{\vec{0},\vec{x}}[i \dots k]$, for $0 \leq i < k \leq \left|\pi_{\vec{0},\vec{x}}\right|$, be the first pumpable segment of $\pi_{\vec{0},\vec{x}}$, and $\vec{u} = \pi_{\vec{0},\vec{x}}[k] - \pi_{\vec{0},\vec{x}}[i]$. Let $\pi^{\rightarrow}_{\pi_{\vec{0},\vec{x}}[i]}$ be the $\vec{u}$-semi-periodic path in G_α originating at $\pi_{\vec{0},\vec{x}}[i]$, and $\pi^{\rightarrow} = \pi_{\vec{0},\vec{x}}[0 \dots i] \cup \pi^{\rightarrow}_{\pi_{\vec{0},\vec{x}}[i]}$.

If $\vec{x} \in \text{dom } \mathcal{F}(\alpha \restriction \pi^{\rightarrow})$ then we are done because $\mathcal{F}(\alpha \restriction \pi^{\rightarrow})$ contains a sub-assembly whose domain is the semi-periodic path $\pi^{\rightarrow}_{\pi_{\vec{0},\vec{x}}[i]}$, which starts in $D_1 \subset D_2$, so assume that $\vec{x} \notin \text{dom } \mathcal{F}(\alpha \restriction \pi^{\rightarrow})$. The path $\pi^{\rightarrow}$ is the base of the comb we will now construct.

Since $\vec{x} \notin \text{dom } \mathcal{F}(\alpha \restriction \pi^{\rightarrow})$ and because there is a simple finite path in G_α from some point $\vec{s}$ on the tail of $\pi^{\rightarrow}$ to $\vec{x}$, Lemma 6.6.8 tells us that there is a $\vec{v}$-semi-periodic path π in G_α originating at $\vec{s}$ with $\pi \cap \pi^{\rightarrow} = \varnothing$. Assume that $\vec{u} \neq z \cdot \vec{v}$ for any $z \in \mathbb{R}$. In this case, let $\pi^{\uparrow} = \pi$. The assembly $\alpha \restriction \pi^{\uparrow}$ is a tooth of some comb with base $\alpha \restriction \pi^{\rightarrow}$. Now define the following assembly.

$$\alpha^* = \alpha \restriction \left(\pi^{\rightarrow} \cup \bigcup_{n \in \mathbb{N}} \left(n \cdot \vec{u} + \pi^{\uparrow} \right) \right).$$

It is clear from the definition that α^* is a comb in α (starting at some point in D_2) with respect to $\pi^{\rightarrow}$ and $\pi^{\uparrow}$. By Lemma 6.6.9, it follows that $\vec{x} \in \mathcal{F}(\alpha^*)$ or α is doubly periodic.

We have shown that, assuming $\vec{u} \neq z \cdot \vec{v}$ for any $z \in \mathbb{R}$, every point $\vec{x} \in X - D_1$ is contained in the finite closure of some comb or periodic path originating at a point in D_2 (unless α is doubly periodic). Furthermore, since D_2 is finite, there are at most a finite number of combs. The theorem follows by Observation 6.6.10.

Recall that π is an eventually $\vec{v}$-semi-periodic path in G_α, and we earlier assumed that $\vec{u} \neq z \cdot \vec{v}$ for any $z \in \mathbb{R}$. Now assume that $\vec{u} = z \cdot \vec{v}$ for some $z \in \mathbb{R}$. Suppose $z < 0$. In this case, let $\pi^{\leftarrow} = \pi$. If $\vec{x} \in \mathcal{F}(\alpha \restriction (\pi^{\rightarrow} \cup \pi^{\leftarrow}))$, then we are done since $\pi^{\leftarrow}$ originates within D_2. If not, then it must be the case that there is a point $\vec{s'}$ on the tail of $\pi^{\leftarrow}$ such that there is a simple finite path from $\vec{s'}$ to $\vec{x}$ in G_α. Then Lemma 6.6.8 tells us that there is an eventually $\vec{w}$-semi-periodic path π' in G_α originating at $\vec{s'}$ with $\pi' \cap (\pi^{\leftarrow} \cup \pi^{\rightarrow}) = \varnothing$. Let $\pi^{\nearrow} = \pi'$.

We now have three eventually periodic paths in G_α, $\pi^{\leftarrow}$, $\pi^{\rightarrow}$, and $\pi^{\nearrow}$, whose respective tails are all disjoint. Moreover, at least two of these paths must either be "parallel," or two of them are neither parallel nor anti-parallel, and the argument assuming $\vec{u} \neq z \cdot \vec{v}$ for any $z \in \mathbb{R}$

applies. If they are parallel, then by Lemma 6.6.5, the base $\pi^{\rightarrow}$ forms a periodic path that cuts the plane into two half-planes. Using reasoning that is similar to the above arguments, it is easy to verify that this implies that α is either

1. the finite closure of a (two-way infinite) periodic path bisecting the plane (hence a semi-linear set),

2. (1) unioned with the finite closure of a comb (with a two-way infinite base) covering one of the half planes formed by the periodic bisection, or

3. (1) unioned with two-way infinite combs on both sides of the bisection, each covering one of the half planes.

In each case, it is clear that the set formed is a semilinear set.

To show the reverse direction, that every semilinear set weakly self-assembles in some temperature 1 tile assembly system (in fact, in a directed, pumpable TAS), let $X = \bigcup_{i=1}^{k} X_i$ be semilinear, with each X_i a linear set. If the "cones" formed by each X_i do not overlap, it is easy to create a TAS that grows hard-coded paths from the seed to each base point of the cone, each of which assembles a comb that weakly assembles the linear set. Consider then if X_i and X_j do have their cones overlap (note that this may not necessarily mean the sets themselves intersect, only that they "interleave").

Then either one cone is contained in the other, or they intersect but each have a region not contained in the other. In either case, the union of the two cones can be partitioned into at most three cones, each of which can be weakly assembled. In the first case, supposing X_i's cone is contained in X_j's cone, the three regions are the portion of X_j to the clockwise direction of X_i, the portion where they intersect, and the portion of X_j to the counterclockwise direction of X_i. The middle portion will need two kinds of black tiles, one for X_i and one for X_j, but in this portion, the set will still be semilinear with periods equal to the least common multiple of the original periods. In the second case, that X_i and X_j have their cones intersect but neither is contained in the other, the three regions are "only X_i", "X_i and X_j", and "only X_j". In

either case we have partitioned the union of two cones into three non-overlapping cones, each of which can be weakly assembled by three distinct combs.

This argument is easily extended to the case that more than two linear sets overlap at once. Since only a finite number of linear sets compose X, we have shown that X can be decomposed into a finite number of non-overlapping sets, each of which can be weakly assembled, and each of which is reachable from the origin without intersecting the others. It follows that X weakly self-assembles at temperature 1. □

Proof. (*Sketch of Theorem 6.5.2*) We modify the temperature 2 ("wedge") construction of [28] as follows. In that construction, a row connects to the row to the north, representing the next configuration of M, via a double bond at the tile representing the tape head, with a tape head "propagation" tile placed to the north (propagating information about the *next* tape head position). Once the tape head propagation tile is in place, the remainder of the row can fill in, copying the bit from the tile to the south, and with the tile immediately to the left or right of the first tile representing the new tape position. In the revised construction, the bottommost row representing the initial configuration contains strength-1 bonds on the north side, allowing the data on the tape to be copied to the next row, even if the tape head propagation tile has not been placed yet. The tiles copied to the north (and subsequent tiles to the north of them) have bonds to the west and east that do not interact (have strength 0 with each other). The tape head propagation tile is designed to have a strength -1 interaction with the west (resp. east) side if the tape head moves left (resp. right). This "unlocks" the vertical column that may have grown over the new tape head position, removing it so that the tape head tile may now be placed there. The glue strengths on the west (resp. east) side have strength -1 with all glues except for the correct tape head tile, and strength 0 with the tape head tile, which, together with the strength 1 bond from the tile to the south, is enough to allow the tape head tile to bind. Other than the old and new tape head positions, all other tiles in a given row are simply copied from the south with no interaction with neighbors to the west or east. To ensure that the tape head propagation tile is not itself disconnected due to its strength -1 bond with its west or east neighbor, it has a strength 2 bond with the tape head tile to its south.

Otherwise, instead of unlocking the vertical column where the tape head is supposed to go, the tape head propagation tile might be removed instead. □

Bibliography

[1] Leonard Adleman, Qi Cheng, Ashish Goel, and Ming-Deh Huang. Running time and program size for self-assembled squares. In *STOC '01: Proceedings of the thirty-third annual ACM Symposium on Theory of Computing*, pages 740–748, New York, NY, USA, 2001. ACM.

[2] Leonard M. Adleman, Jarkko Kari, Lila Kari, Dustin Reishus, and Petr Sosík. The undecidability of the infinite ribbon problem: Implications for computing by self-assembly. *SIAM Journal on Computing*, 38(6):2356–2381, 2009.

[3] Robert D. Barish, Rebecca Schulman, Paul W. Rothemund, and Erik Winfree. An information-bearing seed for nucleating algorithmic self-assembly. *Proceedings of the National Academy of Sciences*, March 2009.

[4] Florent Becker. Pictures worth a thousand tiles, a geometrical programming language for self-assembly. *Theoretical Computer Science*, 410(16):1495–1515, 2009.

[5] Florent Becker, Ivan Rapaport, and Eric Rémila. Self-assembling classes of shapes with a minimum number of tiles, and in optimal time. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 45–56, 2006.

[6] Manuel Blum. On the size of machines. *Information and Control*, 11(3):257–265, 1967.

[7] Daniel M. Bollag. Gel-filtration chromatography. *Methods in Molecular Biology*, 36, November 1994.

[8] Harish Chandran, Nikhil Gopalkrishnan, and John H. Reif. The tile complexity of linear assemblies. In *36th International Colloquium on Automata, Languages and Programming*, volume 5555, 2009.

[9] Ho-Lin Chen, Rebecca Schulman, Ashish Goel, and Erik Winfree. Reducing facet nucleation during algorithmic self-assembly. *Nano Letters*, 7(9):2913–2919, September 2007.

[10] Hui-Hsien Chou, Wei Huang, and James A. Reggia. The Trend cellular automata programming environment. *SIMULATION: Transactions of The Society for Modeling and Simulation International*, 78:5975, 2002.

[11] Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T. Schweller, and Diane L. Souvaine. Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues. *Natural Computing*, 7(3):347–370, 2008.

[12] David Doty. Randomized self-assembly for exact shapes. In *Proceedings of the Fiftieth IEEE Conference on Foundations of Computer Science (FOCS)*, 2009.

[13] David Doty, Jack H. Lutz, Matthew J. Patitz, Scott M. Summers, and Damien Woods. Random number selection in self-assembly. In *Proceedings of The Eighth International Conference on Unconventional Computation (Porta Delgada (Azores), Portugal, September 7-11, 2009)*, 2009.

[14] David Doty and Matthew J. Patitz. A domain specific language for programming in the tile assembly model. In *Proceedings of The Fifteenth International Meeting on DNA Computing and Molecular Programming (Fayetteville, Arkansas, USA, June 8-11, 2009)*, 2009. to appear.

[15] David Doty, Matthew J. Patitz, and Scott M. Summers. Limitations of self-assembly at temperature 1. In *Proceedings of The Fifteenth International Meeting on DNA Computing and Molecular Programming (Fayetteville, Arkansas, USA, June 8-11, 2009)*, 2009. to appear.

[16] Martin Fowler. Language workbenches: The killer-app for domain specific languages?, June 2005. `http://martinfowler.com/articles/languageWorkbench.html`.

[17] Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, January 1978.

[18] Ming-Yang Kao and Robert T. Schweller. Reducing tile complexity for self-assembly through temperature programming. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006), Miami, Florida, Jan. 2006, pp. 571-580*, 2007.

[19] Ming-Yang Kao and Robert T. Schweller. Randomized self-assembly for approximate shapes. In Luca Aceto, Ivan Damgrd, Leslie Ann Goldberg, Magns M. Halldrsson, Anna Inglfsdttir, and Igor Walukiewicz, editors, *International Colloqium on Automata, Languages, and Programming (ICALP)*, volume 5125 of *Lecture Notes in Computer Science*, pages 370–384. Springer, 2008.

[20] Donald E. Knuth. *The Art of Computer Programming, volume 2: Seminumerical Algorithms*. Addison-Wesley, 1997.

[21] James I. Lathrop, Jack H. Lutz, Matthew J. Patitz, and Scott M. Summers. Computability and complexity in self-assembly. In *Proceedings of The Fourth Conference on Computability in Europe (Athens, Greece, June 15-20, 2008)*, 2008.

[22] James I. Lathrop, Jack H. Lutz, and Scott M. Summers. Strict self-assembly of discrete Sierpinski triangles. *Theoretical Computer Science*, 410:384–405, 2009.

[23] Furong Liu, Ruojie Sha, and Nadrian C. Seeman. Modifying the surface features of two-dimensional DNA crystals. *Journal of the American Chemical Society*, 121(5):917–922, 1999.

[24] Chengde Mao, Thomas H. LaBean, John H. Relf, and Nadrian C. Seeman. Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. *Nature*, 407(6803):493–6, 2000.

[25] Chengde Mao, Weiqiong Sun, , and Nadrian C. Seeman. Designed two-dimensional DNA holliday junction arrays visualized by atomic force microscopy. *Journal of the American Chemical Society*, 121(23):5437–5443, 1999.

[26] Michael Mitzenmacher and Eli Upfal. *Probability and Computing*. Cambridge University Press, 2005.

[27] Matthew J. Patitz. Simulation of self-assembly in the abstract tile assembly model with ISU TAS. In *6th Annual Conference on Foundations of Nanoscience: Self-Assembled Architectures and Devices (Snowbird, Utah, USA, April 20-24 2009)*., 2009.

[28] Matthew J. Patitz and Scott M. Summers. Self-assembly of decidable sets. In *Proceedings of The Seventh International Conference on Unconventional Computation (Vienna, Austria, August 25-28, 2008)*, 2008.

[29] Mojżesz Presburger. Ueber die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen. In *welchem die Addition als einzige Operation hervortritt. Compte Rendus du I. Congrks des Mathematiciens des pays Slavs, Warsaw*, pages 92–101, 1930.

[30] John H. Reif, Sudheer Sahu, and Peng Yin. Complexity of graph self-assembly in accretive systems and self-destructible systems. In *DNA*, pages 257–274, 2005.

[31] Paul W. K. Rothemund. *Theory and Experiments in Algorithmic Self-Assembly*. PhD thesis, University of Southern California, December 2001.

[32] Paul W. K. Rothemund and Erik Winfree. The program-size complexity of self-assembled squares (extended abstract). In *STOC '00: Proceedings of the thirty-second annual ACM Symposium on Theory of Computing*, pages 459–468, New York, NY, USA, 2000. ACM.

[33] Paul W.K. Rothemund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology*, 2(12), 2004.

[34] Joseph Sambrook and David Russell. *Molecular Cloning: A Laboratory Manual*. Cold Spring Harbor Laboratory Press, 2001.

[35] Nadrian C. Seeman. Nucleic-acid junctions and lattices. *Journal of Theoretical Biology*, 99:237–247, 1982.

[36] A.L. Shapiro, E. Viñuela, and J.V. Maizel Jr. Molecular weight estimation of polypeptide chains by electrophoresis in SDS-polyacrylamide gels. *Biochem Biophys Res Commun.*, 28:815–820, November 1967.

[37] David Soloveichik and Erik Winfree. Complexity of compact proofreading for self-assembled patterns. In *The eleventh International Meeting on DNA Computing*, 2005.

[38] David Soloveichik and Erik Winfree. Complexity of self-assembled shapes. *SIAM Journal on Computing*, 36(6):1544–1569, 2007.

[39] Antoine Spicher, Olivier Michel, and Jean-Louis Giavitto. Algorithmic self-assembly by accretion and by carving in MGS. In *Artificial Evolution*, volume 3871 of *Lecture Notes in Computer Science*, pages 189–200. Springer, 2005.

[40] Ryan Stansifer. Presburger's article on integer arithmetic: Remarks and translation. Technical Report TR84-639, Cornell University, Computer Science Department, September 1984.

[41] Scott M. Summers. Reducing tile complexity for the self-assembly of scaled shapes through temperature programming. Technical Report 0907.1307, Computing Research Repository, 2009.

[42] John von Neumann. Various techniques for use in connection with random digits. In *von Neumann's Collected Works*, volume 5, pages 768–770. Pergamon, 1963.

[43] Hao Wang. Proving theorems by pattern recognition – II. *The Bell System Technical Journal*, XL(1):1–41, 1961.

[44] Hao Wang. Dominoes and the AEA case of the decision problem. In *Proceedings of the Symposium on Mathematical Theory of Automata (New York, 1962)*, pages 23–55. Polytechnic Press of Polytechnic Inst. of Brooklyn, Brooklyn, N.Y., 1963.

[45] Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.

[46] Erik Winfree. Simulations of computing by self-assembly. Technical Report CaltechC-STR:1998.22, California Institute of Technology, 1998.

[47] Erik Winfree and Renat Bekbolatov. Proofreading tile sets: Error correction for algorithmic self-assembly. In Junghuei Chen and John H. Reif, editors, *DNA*, volume 2943 of *Lecture Notes in Computer Science*, pages 126–144. Springer, 2003.

[48] Erik Winfree, Furong Liu, Lisa A. Wenzler, and Nadrian C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394(6693):539–44, 1998.