IOWA STATE UNIVERSITY
Digital Repository

Graduate Theses and Dissertations

Iowa State University Capstones, Theses and Dissertations

2010

Universality in algorithmic self-assembly

Scott Summers
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/etd

 Part of the Computer Sciences Commons

Recommended Citation
Summers, Scott, "Universality in algorithmic self-assembly" (2010). *Graduate Theses and Dissertations*. 11622.
https://lib.dr.iastate.edu/etd/11622

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

www.manaraa.com

Universality in algorithmic self-assembly

by

Scott Martin Summers

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
Giora Slutzki
John Mayfield
Pavan Aduri

Iowa State University

Ames, Iowa

2010

Copyright © Scott Martin Summers, 2010. All rights reserved.

www.manaraa.com

DEDICATION

I would like to dedicate this thesis to my beautiful (and patient) wife Diana and to my son Owen. They are the source of my inspiration to always work hard and do good in my life. Without them, I am lost and would not have been able to complete this work.

I would also like to thank all of my friends and family for their loving guidance during the writing of this thesis.

www.manaraa.com

TABLE OF CONTENTS

www.manaraa.com

www.manaraa.com

www.manaraa.com

ACKNOWLEDGEMENTS

Initially as a PhD student, you are a lot like an endothermic chemical reaction: a lot of energy in but not a lot out. However, eventually through an "equimolar" combination of hard work, careful guidance, and luck, you become "academically exothermic."

The primary source of my "academic activation energy" was of course provided by my co-major professor Jack Lutz. Throughout my journey as one of his students, Jack has constantly challenged me with interesting research projects and always encouraged me to "work out all the details." Jack was also instrumental in cultivating my love of teaching Computer Science–most notably when he let me lecture for him in his classes and when he gave me the opportunity to serve as the sole instructor for the *theory of computation* in the spring semester of 2008.

Matt Patitz is a person to whom I will forever be indebted. I met Matt in the fall of 2005 at Jack's Christmas party and since then, he has been a hard-working co-author (this is actually a gross understatement), a source of inspiration and most importantly, a true and genuine friend. This thesis would simply not have happened without Matt's friendship and advice. Working with Matt has always been a pleasure and I hope that we can continue to do so throughout our professional careers and well beyond!

This thesis also owes its existence to Damien Woods–a research scientist whom I first met at a conference in the summer of 2008. Since then, I have come to learn first-hand from working with him on several projects that he is simply a world-class researcher and an even better friend. I look forward to continue to work with Damien in the future.

David Doty is another person to whom this thesis owes its existence. Dave has probably forgotten more about theoretical Computer Science than I will ever know. I consider myself lucky to be able to call him not only a collaborator but also a friend.

www.manaraa.com

Although we do not share the same particular research interests, I consider myself extremely lucky to know–and have thoroughly enjoyed interacting with–Satyadev Nandakumar, Xioayang Gu and Brian Patterson.

Finally, I would like to thank Jim Lathrop for serving as my co-major professor.

This thesis was supported by National Science Foundation Grants 0652569 and 0728806, and by NSF-IGERT Training Project in Computational Molecular Biology Grant number DGE-0504304.

www.manaraa.com

ABSTRACT

Tile-based self-assembly is a model of "algorithmic crystal growth" in which square "tiles" represent molecules that bind to each other via specific and variable-strength bonds on their four sides, driven by random mixing in solution but constrained by the local binding rules of the tile bonds. In the late 1990s, Erik Winfree introduced a discrete mathematical model of DNA tile assembly called the abstract Tile Assembly Mode. Winfree proved that the Tile Assembly Model is *computationally universal*, i.e., that any Turing machine can be encoded into a finite set of tile types whose self-assembly simulates that Turing machine. In this thesis, we investigate tile-based self-assembly systems that exhibit *Turing universality*, *geometric universality* and *intrinsic universality*.

We first establish a novel characterization of the computably enumerable languages in terms of self-assembly–the proof of which is a novel proof of the Turing-universality of the Tile Assembly Model in which a particular Turing machine is simulated on all inputs in parallel in the two-dimensional discrete Euclidean plane.

Then we prove that the multiple temperature tile assembly model (introduced by Aggarwal, Cheng, Goldwasser, Kao, and Schweller) exhibits a kind of "geometric universality" in the sense that there is a small (constant-size) universal tile set that can be programmed via deliberate changes in the system temperature to uniquely produce any finite shape.

Just as other models of computation such as the Turing machine and cellular automaton are known to be "intrinsically universal," (e.g., Turing machines can simulate other Turing machines, and cellular automata other cellular automata), we show that tile assembly systems satisfying a natural condition known as *local consistency* are able to simulate other locally consistent tile assembly systems. In other words, we exhibit a particular locally consistent tile

www.manaraa.com

assembly system that can simulate the behavior–as opposed to only the final result–of any other locally consistent tile assembly system.

Finally, we consider the notion of universal fault-tolerance in algorithmic self-assembly with respect to the *two-handed* Tile Assembly Model, in which large aggregations of tiles may attach to each other, in contrast to the *seeded* Tile Assembly Model, in which tiles aggregate one at a time to a single specially-designated "seed" assembly. We introduce a new model of fault-tolerance in self-assembly: the *fuzzy temperature* model of faults having the following informal characterization: the system temperature is normally 2, but may drift down to 1, allowing unintended temperature-1 growth for an arbitrary period of time. Our main construction, which is a tile set capable of uniquely producing an $n \times n$ square with $O(\log n)$ unique tile types in the fuzzy temperature model, is not universal but presents novel technique that we hope will ultimately pave the way for a "universal fuzzy-fault-tolerant" tile assembly system in the future.

www.manaraa.com

CHAPTER 1. Introduction

1.1 Motivation

Tile-based self-assembly is a model of "algorithmic crystal growth" in which square "tiles" represent molecules that bind to each other via specific and variable-strength bonds on their four sides, driven by random mixing in solution but constrained by the local binding rules of the tile bonds. Beginning with the experimental work of Seeman in the early 1980s [51], such molecules have been engineered from DNA in the laboratory (see Figures 1.1 and 1.2), and used to create a variety of sophisticated self-assembled structures [6, 13, 34, 36, 37, 48, 62] such as Sierpinski triangles and binary counters.

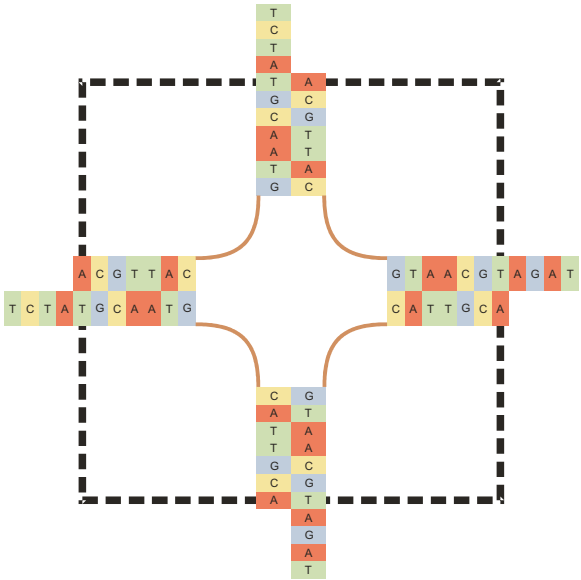


Figure 1.1: Over-simplified version of a DNA tile. The "sticky ends" bind with their complements.

www.manaraa.com

In the late 1990s, Winfree [58] introduced a mathematical model of DNA tile assembly called the abstract Tile Assembly Mode. This model, which was soon refined by Rothemund and Winfree [46,47], is an extension of Wang tiling [56,57]. (see also [1,44,54].) The Tile Assembly Model, which is described in chapter 2, uses square tiles with various types and strengths of "glue" on their edges as abstractions of DNA tiles adsorbing to a growing structure on a very flat surface such as mica. Winfree [58] proved that the Tile Assembly Model is *computationally universal*, i.e., that any Turing machine can be encoded into a finite set of tile types whose self-assembly simulates that Turing machine.

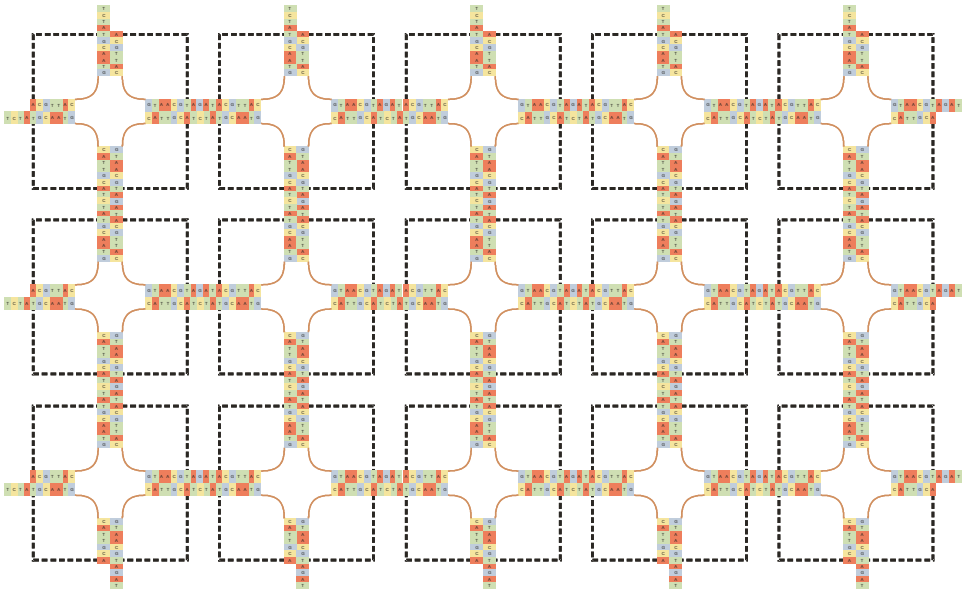


Figure 1.2: Self-assembly of a regular array with DNA tiles.

However, is tile-based self-assembly capable of stronger notions of universality? In other words, is it possible for tile assembly systems to simulate the behavior of other tile assembly systems directly? And if so, then what are the complexities of such universal tile assembly systems and how does one go about programming universal self-assembly systems? These are precisely the questions that we address in this thesis.

www.manaraa.com

1.2 Universality in Algorithmic Self-Assembly

What follows is an informal description of the results that are presented in this thesis.

1.2.1 Turing Universality in the Abstract Tile Assembly Model

The first chapter of this thesis explores the impact of geometry on computability and complexity in Winfree's abstract model of nanoscale tile self-assembly. The first main theorem of this chapter says that there is a roughly quadratic function f such that a set $A \subseteq \mathbb{Z}^+$ is computably enumerable if and only if the set $X_A = \{(f(n), 0) \mid n \in A\}$ – a simple representation of A as a set of points on the x-axis – self-assembles in Winfree's sense. In contrast, the second main theorem of this chapter says that there are decidable sets $D \subseteq \mathbb{Z} \times \mathbb{Z}$ that do *not* self-assemble in Winfree's sense.

The first main theorem of this chapter is established by an explicit translation of an arbitrary Turing machine M to a modular tile assembly system $\mathcal{T}_M$, together with a proof that $\mathcal{T}_M$ carries out concurrent simulations of M on all positive integer inputs.

1.2.2 Geometric Universality in the Multiple Temperature Model

The second chapter concerns the self-assembly of scaled-up versions of arbitrary finite shapes in the multiple temperature model that was introduced by Aggarwal, Cheng, Goldwasser, Kao, and Schweller [14]. The multiple temperature model is a natural generalization of Winfree's abstract tile assembly model, where the temperature of a tile system is allowed to be shifted up and down as self-assembly proceeds. We first exhibit two *geometrically universal* (and therefore constant-size) tile sets in which scaled-up versions of arbitrary shapes self-assemble. Our first tile set has the property that each scaled shape self-assembles via an asymptotically "Kolmogorov-optimum" temperature sequence but the scaling factor grows with the size of the shape being assembled. In contrast, our second tile set assembles each scaled shape via a temperature sequence whose length is proportional to the number of points in the shape but the scaling factor is a constant independent of the shape being assembled. We then show that there is no constant-size tile set that can uniquely assemble an arbitrary

www.manaraa.com

(non-scaled, connected) shape in the multiple temperature model, i.e., the scaling is necessary for self-assembly. This answers an open question of Kao and Schweller [26], who asked whether such a tile set exists.

1.2.3 Intrinsic Universality in the Abstract Tile Assembly Model

In the third chapter, we show that the abstract Tile Assembly Model exhibits a strong notion of universality where the goal is to give a single tile assembly system that simulates the behavior of any other tile assembly system. We give a tile assembly system that is capable of simulating a very wide class of tile systems, including itself. Specifically, we give a tile set that simulates the assembly of any tile assembly system in a class of systems that we call *locally consistent*: each tile binds with exactly the strength needed to stay attached, and that there are no glue mismatches between tiles in any produced assembly.

Our construction is reminiscent of the studies of *intrinsic universality* of cellular automata by Ollinger and others [4, 21, 38–40], in the sense that our simulation of a tile system T by a tile system U represents each tile in an assembly produced by T by a $c \times c$ block of tiles in U, where c is a constant depending on T but not on the size of the assembly T produces (which may in fact be infinite). Also, our construction improves on earlier simulations of tile assembly systems by other tile assembly systems in that we simulate the actual process of self-assembly, not just the end result, as in Soloveichik and Winfree's construction [54], and we do not discriminate against infinite structures. Both of the previously mentioned results simulate only temperature 1 systems, whereas our construction simulates tile assembly systems operating at temperature 2.

1.2.4 Toward Fault-Tolerant Universality in the Fuzzy Temperature Model

In the final chapter, we consider the problem of fault-tolerance in nanoscale algorithmic self-assembly. We employ a variant of Winfree's abstract Tile Assembly Model, the *two-handed*, in which square "tiles" – a model of molecules constructed from DNA for the purpose of engineering self-assembled nanostructures – aggregate according to specific binding sites of

www.manaraa.com

varying strengths, and in which large aggregations of tiles may attach to each other, in contrast to the *seeded*, in which tiles aggregate one at a time to a single specially-designated "seed" assembly. We focus on a major cause of errors in tile-based self-assembly: that of unintended growth due to "weak" strength-1 bonds, which if allowed to persist, may be stabilized by subsequent attachment of neighboring tiles in the sense that at least energy 2 is now required to break apart the resulting assembly; i.e., the errant assembly is *stable at temperature 2*.

We study a common self-assembly benchmark problem, that of assembling an $n \times n$ square using $O(\log n)$ unique tile types, under the two-handed model of self-assembly. Our main result achieves a much stronger notion of fault-tolerance than those achieved previously [10, 11, 13, 42, 59, 61]. *Arbitrary* strength-1 growth is allowed; however, any assembly that grows sufficiently to become stable at temperature 2 is guaranteed to assemble into the correct final assembly of an $n \times n$ square. In other words, errors due to insufficient attachment, which is the cause of errors studied in earlier papers on fault-tolerance, are prevented *absolutely* in our main construction, rather than only with high probability and for sufficiently small structures, as in previous fault-tolerance studies. We term this the *fuzzy temperature* model of faults, due to the following equivalent characterization: the temperature is normally 2, but may drift down to 1, allowing unintended temperature-1 growth for an arbitrary period of time. Our construction ensures that this unintended growth cannot lead to permanent errors, so long as the temperature is eventually raised back to 2. Thus, our construction overcomes a major cause of errors, insufficient strength-1 attachments becoming stabilized by subsequent growth, without requiring the detachment of strength-2 bonds that slows down previous constructions, and without requiring the careful fine-tuning of thermodynamic parameters to balance forward and reverse rates of reaction necessary in earlier work on fault-tolerance.

Although we focus on the task of assembling an $n \times n$ square, our construction uses a number of geometric motifs and synchronization primitives that will likely prove useful in other universality constructions.

www.manaraa.com

CHAPTER 2. Preliminaries

2.1 Notation

We work in the discrete Euclidean plane $\mathbb{Z}^2 = \mathbb{Z} \times \mathbb{Z}$. We write U_2 for the set of all *unit vectors*, i.e., vectors of length 1, in $\mathbb{Z}^2$. We regard the four elements of U_2 as (names of the cardinal) *directions* 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*. A *cut* of a graph $G = (V, E)$ is a partition $C = (C_0, C_1)$ of V into two nonempty, disjoint subsets C_0 and C_1.

A *binding function* on a graph $G = (V, E)$ is a function $\beta : E \to \mathbb{N}$. (Intuitively, if $\{u, v\} \in E$, then $\beta(\{u, v\})$ is the strength with which u is bound to v by $\{u, v\}$ according to β. If β is a binding function on a graph $G = (V, E)$ and $C = (C_0, C_1)$ is a cut of G, then the *binding strength* of β on C is

$$\beta_C = \{\beta(e) \mid e \in E, e \cap C_0 \neq \varnothing, \text{ and } e \cap C_1 \neq \varnothing\}.$$

The *binding strength* of β on the graph G is then

$$\beta(G) = \min\{\beta_C \mid C \text{ is a cut of } G\}.$$

A *binding graph* is an ordered triple $G = (V, E, \beta)$, where (V, E) is a graph and β is a binding function on (V, E). If $\tau \in \mathbb{N}$, then a binding graph $G = (V, E, \beta)$ is *τ-stable* if $\beta(V, E) \geq \tau$.

A *grid graph* is a graph $G = (V, E)$ in which $V \subseteq \mathbb{Z}^2$ and every edge $\{\vec{m}, \vec{n}\} \in E$ has the property that $\vec{m} - \vec{n} \in U_2$. The *full grid graph* on a set $V \subseteq \mathbb{Z}^2$ is the graph $G_V^{\#} = (V, E)$ in which E contains *every* $\{\vec{m}, \vec{n}\} \in [V]^2$ such that $\vec{m} - \vec{n} \in U_2$.

www.manaraa.com

A *shape* is a set $X \subseteq \mathbb{Z}^2$ such that $G_X^{\#}$ is connected. In this paper, we consider scaled-up versions of finite shapes. Formally, if X is a shape and $c \in \mathbb{N}$, then a *c-scaling* of X is defined as the set $X^c = \left\{(x,y) \in \mathbb{Z}^2 \;\middle|\; \left(\left\lfloor \frac{x}{c} \right\rfloor, \left\lfloor \frac{y}{c} \right\rfloor\right) \in X \right\}$. Intuitively, X^c is the shape obtained by replacing each point in X with a $c \times c$ block of points. We refer to the natural number c as the *scaling factor* or *resolution loss*. Note that scaled shapes have been studied extensively in the context of a variety of self-assembly systems [10, 16, 54, 61].

We say that f is a *partial function* from a set X to a set Y, and we write $f : X \dashrightarrow Y$, if $f : D \to Y$ for some set $D \subseteq X$. In this case, D is the *domain* of f, and we write $D = \text{dom } f$.

All logarithms here are base-2.

2.2 The Multiple Temperature Tile Assembly Model

We formally develop the ideas behind the Multiple Temperature Tile Assembly Model. Our development largely follows that of [26], but some of our terminology and notation are specifically tailored to our objectives. In particular, our version of the model bestows equal status on finite and infinite assemblies.

Definition. A *tile type* over an alphabet Σ is a function $t : U_2 \to \Sigma^* \times \mathbb{N}$. We write $t = (\text{col}_t, \text{str}_t)$, where $\text{col}_t : U_2 \to \Sigma^*$, and $\text{str}_t : U_2 \to \mathbb{N}$ are defined by $t(\vec{u}) = (\text{col}_t(\vec{u}), \text{str}_t(\vec{u}))$ for all $\vec{u} \in U_2$.

Intuitively, a tile of type t is a unit square. It can be translated but not rotated, so it has a well-defined "side $\vec{u}$" for each $\vec{u} \in U_2$. Each side $\vec{u}$ of the tile is covered with a "glue" of *color* $\text{col}_t(\vec{u})$ and *strength* $\text{str}_t(\vec{u})$. If tiles of types t and t' are placed with their centers at $\vec{m}$ and $\vec{m} + \vec{u}$, respectively, where $\vec{m} \in \mathbb{Z}^2$ and $\vec{u} \in U_2$, then they will *bind* with strength $\text{str}_t(\vec{u}) \cdot [\![t(\vec{u}) = t'(-\vec{u})]\!]$ where $[\![\phi]\!]$ is the *Boolean* value of the statement ϕ. Note that this binding strength is 0 unless the adjoining sides have glues of both the same color and the same strength.

For the remainder of this section, unless otherwise specified, T is an arbitrary set of tile types, and $\tau \in \mathbb{N}$ is the "temperature."

www.manaraa.com

Definition. A T-*configuration* is a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$.

Intuitively, a configuration is an assignment α in which a tile of type $\alpha(\vec{m})$ has been placed (with its center) at each point $\vec{m} \in \text{dom } \alpha$. The following data structure characterizes how these tiles are bound to one another.

Definition. The *binding graph of* a T-configuration $\alpha : \mathbb{Z}^2 \dashrightarrow T$ is the binding graph $G_\alpha = (V, E, \beta)$, where (V, E) is the grid graph given by $V = \text{dom } \alpha$, and $\{\vec{m}, \vec{n}\} \in E$ if and only if

1. $\vec{m} - \vec{n} \in U_n$,
2. $\text{col}_{\alpha(\vec{m})}(\vec{n} - \vec{m}) = \text{col}_{\alpha(\vec{n})}(\vec{m} - \vec{n})$, and
3. $\text{str}_{\alpha(\vec{m})}(\vec{n} - \vec{m}) > 0$.

The binding function $\beta : E \to \mathbb{Z}^+$ is given by

$$\beta(\{\vec{m}, \vec{n}\}) = \text{str}_{\alpha(\vec{m})}(\vec{n} - \vec{m})$$

for all $\{\vec{m}, \vec{n}\} \in E$.

Definition.

1. A T-configuration α is τ-*stable* if its binding graph G_α is τ-stable.
2. A T-configuration α is *connected* if its binding graph G_α is connected. We write $\mathcal{A}_T$ for the set of all connected T-configurations.
3. A τ-T-*assembly* is a T-configuration that is τ-stable. We write $\mathcal{A}_T^\tau$ for the set of all τ-T-assemblies.

Definition. Let α and α' be T-configurations.

1. α is a *subconfiguration* of α', and we write $\alpha \sqsubseteq \alpha'$, if $\text{dom } \alpha \subseteq \text{dom } \alpha'$ and, for all $\vec{m} \in \text{dom } \alpha$, $\alpha(\vec{m}) = \alpha'(\vec{m})$.
2. α' is a *single-tile extension* of α if $\alpha \sqsubseteq \alpha'$ and $\text{dom } \alpha' - \text{dom } \alpha$ is a singleton set. In this case, we write $\alpha' = \alpha + (\vec{m} \mapsto t)$, where $\{\vec{m}\} = \text{dom } \alpha' - \text{dom } \alpha$ and $t = \alpha'(\vec{m})$.

www.manaraa.com

3. α' is a *multi-tile deletion* of α if $\alpha' \sqsubseteq \alpha$ and $\text{dom } \alpha - \text{dom } \alpha' = A$, where $A \subseteq Z^2$. In this case, we write $\alpha' = \alpha - A$.

Note that the expression $\alpha + (\vec{m} \mapsto t)$ is only defined when $\vec{m} \in \mathbb{Z}^2 - \text{dom } \alpha$. Likewise, the expression $\alpha' = \alpha - A$ is only defined when $A \in \text{dom } \alpha'$.

We next define the "τ-t-frontier" of a τ-T-assembly α to be the set of all positions at which a tile of type t can be "τ-stably added" to the assembly α.

Definition. Let $\alpha \in \mathcal{A}_T$.

1. For each $t \in T$, the *τ-t-frontier* of α is the set
$$\partial_t^\tau \alpha = \left\{ \vec{m} \in \mathbb{Z}^2 - \text{dom } \alpha \;\middle|\; \sum_{\vec{u} \in U_2} \text{str}_t(\vec{u}) \cdot [\![\alpha(\vec{m} + \vec{u})(-\vec{u}) = t(\vec{u})]\!] \geq \tau \right\}.$$
2. The *τ-frontier* of α is the set
$$\partial^\tau \alpha = \bigcup_{t \in T} \partial_t^\tau \alpha.$$

The following lemma shows that the definition of $\partial_t^\tau \alpha$ achieves the desired effect.

Lemma 1. Let $\alpha \in \mathcal{A}_T^\tau$, $\vec{m} \in \mathbb{Z}^2 - \text{dom } \alpha$, and $t \in T$. Then $\alpha + (\vec{m} \mapsto t) \in \mathcal{A}_T^\tau$ if and only if $\vec{m} \in \partial_t^\tau \alpha$.

The following definition is the set of all (finite or infinite) "portions" of a T-configuration α that will eventually melt away.

Definition. Let α be a T-configuration. Define the *τ-anti-frontier* of α as the set
$$\textit{anti-}\partial^\tau \alpha = \{ B \subseteq \text{dom } \alpha \mid (\text{dom } \alpha - B, B) \text{ is a cut of } G_\alpha = (V, E, \beta) \text{ and } \beta(B) < \tau \}.$$

Notation 1. We write $\alpha \underset{\tau,T}{\longrightarrow} \alpha'$ (or, when τ and T are clear from context, $\alpha \to \alpha'$) to indicate one of the following conditions.

1. $\alpha, \alpha' \in \mathcal{A}_T$ and α' is a single-tile extension of α, or
2. $\text{dom } \alpha - \text{dom } \alpha' \in \textit{anti-}\partial^\tau \alpha$ and α' is a multi-tile deletion of α.

www.manaraa.com

In general, self-assembly occurs with tiles absorbing (or detaching) nondeterministically and asynchronously to a growing (or shrinking) assembly. We now define assembly sequences, which are particular "execution traces" of how this might occur.

Definition. A *τ-T-assembly sequence* is a sequence $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ over $\mathcal{A}_T$, where $k \in \mathbb{Z}^+ \cup \{\infty\}$ such that, for each $i \in \mathbb{N}$ with $1 \leq i+1 < k$, $\alpha_i \xrightarrow[\tau,T]{} \alpha_{i+1}$.

Note that assembly sequences may be finite or infinite in length.

Definition. Let $\alpha \in \mathcal{A}_T$ and $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ be a τ-T-assembly sequence. We say that $\vec{\alpha}$ *converges to* α if there exists $i \in \mathbb{N}$ such that for all $j \geq i$, $\alpha_j \sqsubseteq \alpha$. Moreover, the *result* of a convergent assembly sequence $\vec{\alpha}$ is the unique T-configuration $\alpha = \text{res}(\vec{\alpha})$ satisfying $\text{dom}\, \alpha = \bigcup_{j \leq i < k} \text{dom}\, \alpha_i$ and $\alpha_i \sqsubseteq \alpha$ for each $j \leq i < k$.

Note that it only makes sense for convergent assembly sequences to have results. It is easy to construct an example of an assembly sequence that does not converge and therefore has no clear result. Unless explicitly stated otherwise, all subsequent assembly sequences are assumed to be convergent and therefore have results.

Definition. Let $\alpha, \alpha' \in \mathcal{A}_T$.

1. A *τ-T-assembly sequence from α to α'* is a τ-T-assembly sequence $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ such that $\alpha_0 = \alpha$ and $\text{res}(\vec{\alpha}) = \alpha'$.
2. We write $\alpha \xrightarrow[\tau,T]{} \alpha'$ (or, when τ and T are clear from context, $\alpha \longrightarrow \alpha'$) to indicate that there exists a τ-T-assembly sequence from α to α'.

Definition. An assembly $\alpha \in \mathcal{A}_T^\tau$ is *terminal* if $\partial^\tau \alpha = \varnothing$.

We now define multiple temperature tile assembly systems. Unless otherwise specified, from this point on, $0 < m \in \mathbb{N}$.

Definition.

1. A *generalized tile assembly system* (*GTAS*) is an ordered triple

$$\mathcal{T} = \left(T, \sigma, \langle \tau_j \rangle_{j=0}^{m-1}\right),$$

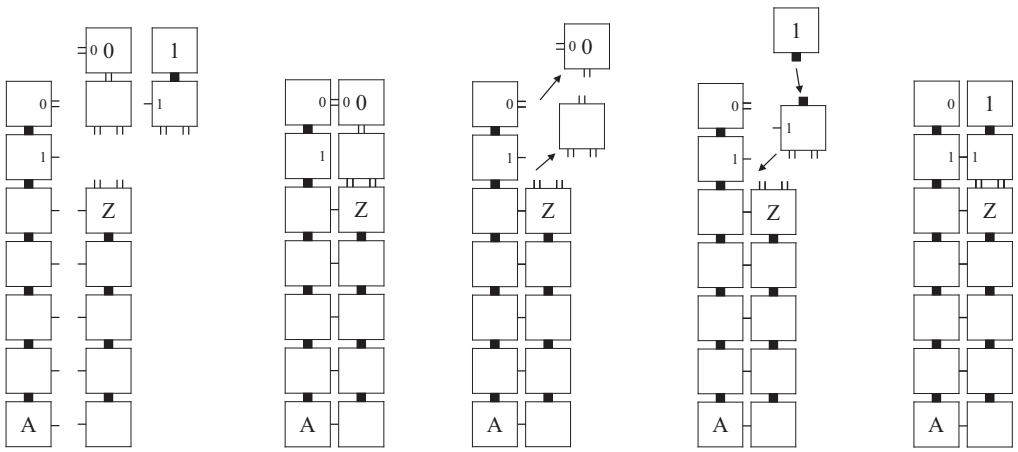
www.manaraa.com

where T is a set of tile types, $\sigma \in \mathcal{A}_T^\tau$ is the *seed assembly*, $0 < m \in \mathbb{N}$ and for each $0 \leq j < m$, $\tau_j \in \mathbb{N}$ is the j^{th}-*temperature*.

2. A *tile assembly system* (*TAS*) is a GTAS $\mathcal{T} = \left(T, \sigma, \langle \tau_j \rangle_{j=0}^{m-1}\right)$ in which the sets T and dom σ are finite.

Notation 2. If $\mathcal{T} = \left(T, \sigma, \langle \tau_j \rangle_{j=0}^{m-1}\right)$ is a GTAS with $m = 1$, then we simply write $\mathcal{T} = (T, \sigma, \tau)$, where $\tau = \tau_0$.

Intuitively, self-assembly in $\mathcal{T} = \left(T, \sigma, \langle \tau_j \rangle_{j=0}^{m-1}\right)$ is carried out in m phases. In the first temperature phase, tiles are (only) added to the existing assembly as they normally would be in the abstract model until a τ_0-stable terminal assembly is reached. In phase two, tiles can accrete to the existing assembly if they can do so with at least strength τ_1. Also, and at any time during the second temperature phase, if there is ever a cut of the assembly having a strength less than τ_1, then all of the tiles on the side of the cut not containing the seed can be removed from the assembly. When a τ_1-stable terminal assembly is reached in phase two, phase three begins and proceeds in a similar fashion. This process continues through the final temperature phase in which tiles are added or removed with respect to the temperature τ_{m-1} until reaching a τ_{m-1}-stable terminal assembly. This notion is made precise as follows.



(a) Tile types (b) $\tau_0 = 2$ (c) $\tau_1 = 5$ (d) (e)

Figure 2.1: In this example, thick notches are strength 5, and thin notches are strength 1.

www.manaraa.com

Definition. Let $\mathcal{T} = \left(T, \sigma, \langle \tau_j \rangle_{j=0}^{m-1}\right)$ be a GTAS. An *assembly sequence for the j^{th} temperature phase of $\mathcal{T}$*, with $0 \leq j < m$, is a sequence $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ over $\mathcal{A}_T$ with $k \in \mathbb{Z}^+ \cup \{\infty\}$ such that $\alpha_0 = \sigma$, for all $i_{j'-1} \leq i < i_{j'}$, $\sigma \sqsubseteq \alpha_i$ and there exists $i_{-1}, i_0, \ldots, i_{j-1} \in \mathbb{Z}^+$, $i_j \in \mathbb{Z}^+ \cup \{\infty\}$ satisfying $0 = i_{-1} < i_0 < i_1 < \cdots < i_{j-1} < i_j = k$ and for all $0 \leq j' \leq j$,

$$\vec{\alpha}_{j'} = \left(\alpha_i \mid i_{j'-1} \leq i < i_{j'}\right)$$

is a $\tau_{j'}$-T-assembly sequence satisfying $\alpha_{i_{j'-1}} \in \mathcal{A}_T^{\tau_{j'}}$ and $\partial^{\tau_{j'}} \text{res}(\vec{\alpha}_{j'}) = \varnothing$. We write $\text{res}(\vec{\alpha}) = \alpha = \text{res}(\vec{\alpha}_j)$ for the *result* of $\vec{\alpha}$.

Definition. Let $\mathcal{T} = \left(T, \sigma, \langle \tau_j \rangle_{j=0}^{m-1}\right)$ be a GTAS. An *assembly sequence in $\mathcal{T}$* is an assembly sequence for the j^{th} temperature phase of $\mathcal{T}$ for some $0 \leq j < m$.

Notation 3. Let $\mathcal{T} = \left(T, \sigma, \langle \tau_j \rangle_{j=0}^{m-1}\right)$ be a GTAS and $\alpha \in \mathcal{A}_T$. If there exists an assembly sequence $\vec{\alpha}$ for the j^{th} temperature phase of $\mathcal{T}$ such that $\text{res}(\vec{\alpha}) = \alpha$, then we write $\sigma \xrightarrow{j} \alpha$.

Definition. Let $\mathcal{T} = \left(T, \sigma, \langle \tau_j \rangle_{j=0}^{m-1}\right)$ be a GTAS.

1. The *set of producible assemblies of the j^{th}* temperature phase of $\mathcal{T}$ is the set
$$\mathcal{A}^j[\mathcal{T}] = \left\{\alpha \in \mathcal{A}_T \;\middle|\; \sigma \xrightarrow{j} \alpha\right\}.$$
2. The *set of terminal assemblies of the j^{th}* temperature phase of $\mathcal{T}$ is the set
$$\mathcal{A}_\square^j[\mathcal{T}] = \left\{\alpha \in \mathcal{A}^j[\mathcal{T}] \;\middle|\; \alpha \in \mathcal{A}_T^{\tau_j} \text{ and } \partial^{\tau_j} \alpha = \varnothing\right\}.$$
3. The *set of terminal assemblies of $\mathcal{T}$* is the set
$$\mathcal{A}_\square[\mathcal{T}] = \mathcal{A}_\square^{m-1}[\mathcal{T}].$$

Definition. Let $\mathcal{T} = \left(T, \sigma, \langle \tau_j \rangle_{j=0}^{m-1}\right)$ be a GTAS.

1. We say that $\mathcal{T}$ *uniquely produces an assembly* $\alpha \in \mathcal{A}_T$ if $\mathcal{A}_\square[\mathcal{T}] = \{\alpha\}$ and for all $0 \leq j < m-1$, $\mathcal{A}_\square^j[\mathcal{T}] = \varnothing$.
2. We say that $\mathcal{T}$ *uniquely produces a shape* $X \subseteq \mathbb{Z}^2$ if $\mathcal{T}$ uniquely produces $\alpha \in \mathcal{A}_T$ and $\text{dom}\, \alpha = X$.

www.manaraa.com

We are primarily interested in the self-assembly of sets.

Definition. Let $\mathcal{T}$ be a GTAS, and let $X \subseteq \mathbb{Z}^2$.

1. The set X *weakly self-assembles* in $\mathcal{T}$ if there is a set $B \subseteq T$ such that, for all $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$, $\alpha^{-1}(B) = X$.

2. The set X *strictly self-assembles* in $\mathcal{T}$ if, for all $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$, dom $\alpha = X$.

Intuitively, a set X weakly self-assembles in $\mathcal{T}$ if there is a designated set B of "black" tile types such that every terminal assembly of $\mathcal{T}$ "paints the set X - and only the set X - black". In contrast, a set X strictly self-assembles in $\mathcal{T}$ if every terminal assembly of $\mathcal{T}$ has tiles on the set X and only on the set X. Clearly, every set that strictly self-assembles in a GTAS $\mathcal{T}$ also weakly self-assembles in $\mathcal{T}$.

We now have the machinery to say what it means for a set in the discrete Euclidean plane to self-assemble in either the weak or the strict sense.

Definition. Let $X \subseteq \mathbb{Z}^2$.

1. The set X *weakly self-assembles* if there is a TAS $\mathcal{T}$ such that X weakly self-assembles in $\mathcal{T}$.

2. The set X *strictly self-assembles* if there is a TAS $\mathcal{T}$ such that X strictly self-assembles in $\mathcal{T}$.

Note that $\mathcal{T}$ is required to be a TAS, i.e., finite, in both parts of the above definition.

www.manaraa.com

CHAPTER 3. Turing Universality in the Abstract Tile Assembly Model

This chapter is joint work with Jack Lutz, James Lathrop and Matthew Patitz and was originally published as [31].

3.1 Introduction

The computational universality of the Tile Assembly Model implies that self-assembly can be algorithmically directed, and hence that a very rich set of structures can be formed by self-assembly. However, as we shall see, this computational universality does not seem to imply a simple characterization of the class of structures that can be formed by self-assembly. The difficulty is that self-assembly (like sensor networks, smart materials, and other topics of current research [5, 7]) is a phenomenon in which the *spatial* aspect of computing plays a crucial role. Two processes, namely self-assembly and the computation that directs it, must take place in the same space and time.

This chapter presents two theorems on the interplay between geometry and computation in the abstract tile self-assembly model (e.g., the multiple temperature model with one temperature phase). To explain our first main theorem, define the function $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ by

$$f(n) = \binom{n+1}{2} + (n+1)\lfloor \log n \rfloor + 6n - 2^{1+\lfloor \log(n) \rfloor} + 2.$$

Note that f is a reasonably simple, strictly increasing, roughly quadratic function of n. For each set $A \subseteq \mathbb{Z}^+$, the set

$$X_A = \{(f(n), 0) \mid n \in A\}$$

is thus a straightforward representation of A as a set of points on the positive x-axis.

www.manaraa.com

The first theorem of this chapter says that *every* computably enumerable set A of positive integers (decidable or otherwise) weakly self-assembles in the sense that there is a tile assembly system $\mathcal{T}_A$ in which the representation X_A self-assembles. Conversely, the existence of such a tile assembly system implies the computable enumerability of A.

In contrast, the second theorem of this chapter says that there are *decidable* sets $D \subseteq \mathbb{Z}^2$ that do *not* weakly self-assemble in any tile assembly system. In fact, we exhibit such a set D for which the condition $(m, n) \in D$ is decidable in time polynomial in $|m| + |n|$.

Taken together, the two theorems in this chapter indicate that the interaction between geometry and computation in self-assembly is not at all simple. Further investigation of this interaction will improve our understanding of tile self-assembly and, more generally, spatial computation.

The proof of the first theorem in this chapter has two features that may be useful in future investigations. First, we give an explicit transformation (essentially a compiler, implemented in C++) of an arbitrary Turing machine M to a tile assembly system $\mathcal{T}_M$ whose self-assembly carries out concurrent simulations of M on (the binary representation of) all positive integer inputs. Second, we prove two lemmas – a pseudoseed lemma and a multiseed lemma – that enable us to reason about tile assembly systems in a modular fashion. This modularity, together with the local determinism method of Soloveichik and Winfree [54], enables us to prove the correctness of $\mathcal{T}_M$.

3.2 Local Determinism

In general, even a directed TAS may have a very large (perhaps uncountably infinite) number of different assembly sequences leading to its terminal assembly. This seems to make it very difficult to prove that a TAS is directed. Fortunately, Soloveichik and Winfree [54] have recently defined a property, *local determinism*, of assembly sequences and proven the remarkable fact that, if a TAS $\mathcal{T}$ has *any* assembly sequence that is locally deterministic, then $\mathcal{T}$ is directed. In what follows, we will review the local determinism method of Soloveichik and Winfree.

www.manaraa.com

Notation 4. For each T-configuration α, each $\vec{m} \in \mathbb{Z}^2$, and each $\vec{u} \in U_2$,

$$\text{str}_\alpha(\vec{m}, \vec{u}) = \text{str}_{\alpha(\vec{m})}(\vec{u}) \cdot [\![\alpha(\vec{m})(\vec{u}) = \alpha(\vec{m} + \vec{u})(-\vec{u})]\!].$$

(The Boolean value on the right is 0 if $\{\vec{m}, \vec{m} + \vec{u}\} \not\subseteq \text{dom } \alpha$.)

Notation 5. If $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ is a τ-T-assembly sequence and $\vec{m} \in \mathbb{Z}^2$, then the $\vec{\alpha}$-*index* of $\vec{m}$ is

$$i_{\vec{\alpha}}(\vec{m}) = \min\{i \in \mathbb{N} \mid \vec{m} \in \text{dom } \alpha_i\}.$$

Observation 2. $\vec{m} \in \text{dom res}(\vec{\alpha}) \Leftrightarrow i_{\vec{\alpha}}(\vec{m}) < \infty$.

Notation 6. If $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ is a τ-T-assembly sequence, then, for $\vec{m}, \vec{m}' \in \mathbb{Z}^2$,

$$\vec{m} \prec_{\vec{\alpha}} \vec{m}' \Leftrightarrow i_{\vec{\alpha}}(\vec{m}) < i_{\vec{\alpha}}(\vec{m}').$$

Definition. (Soloveichik and Winfree [54]) Let $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ be a τ-T-assembly sequence, and let $\alpha = \text{res}(\vec{\alpha})$. For each location $\vec{m} \in \text{dom } \alpha$, define the following sets of directions.

1. $\text{IN}^{\vec{\alpha}}(\vec{m}) = \left\{\vec{u} \in U_2 \;\middle|\; \vec{m} + \vec{u} \prec_{\vec{\alpha}} \vec{m} \text{ and } \text{str}_{\alpha_{i_{\vec{\alpha}}(\vec{m})}}(\vec{m}, \vec{u}) > 0\right\}$.
2. $\text{OUT}^{\vec{\alpha}}(\vec{m}) = \left\{\vec{u} \in U_2 \;\middle|\; -\vec{u} \in \text{IN}^{\vec{\alpha}}(\vec{m} + \vec{u})\right\}$.

Intuitively, $\text{IN}^{\vec{\alpha}}(\vec{m})$ is the set of sides on which the tile at $\vec{m}$ initially binds in the assembly sequence $\vec{\alpha}$, and $\text{OUT}^{\vec{\alpha}}(\vec{m})$ is the set of sides on which this tile propagates information to future tiles.

Note that $\text{IN}^{\vec{\alpha}}(\vec{m}) = \varnothing$ for all $\vec{m} \in \alpha_0$.

Notation 7. If $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ is a τ-T-assembly sequence, $\alpha = \text{res}(\vec{\alpha})$, and $\vec{m} \in \text{dom } \alpha - \text{dom } \alpha_0$, then

$$\vec{\alpha} \setminus \vec{m} = \alpha \upharpoonright \left(\text{dom } \alpha - \{\vec{m}\} - \left(\vec{m} + \text{OUT}^{\vec{\alpha}}(\vec{m})\right)\right).$$

(Note that $\vec{\alpha} \setminus \vec{m}$ is a T-configuration that may or may not be a τ-T-assembly.

Definition. (Soloveichik and Winfree [54]). A τ-T-assembly sequence $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ with result α is *locally deterministic* if it has the following three properties.

www.manaraa.com

1. For all $\vec{m} \in \text{dom } \alpha - \text{dom } \alpha_0$,

$$\sum_{\vec{u} \in \text{IN}^{\vec{\alpha}}(\vec{m})} \text{str}_{\alpha_{i_{\vec{\alpha}}(\vec{m})}}(\vec{m}, \vec{u}) = \tau.$$

2. For all $\vec{m} \in \text{dom } \alpha - \text{dom } \alpha_0$ and all $t \in T - \{\alpha(\vec{m})\}$, $\vec{m} \notin \partial_t^\tau(\vec{\alpha} \setminus \vec{m})$.

3. $\partial^\tau \alpha = \varnothing$.

That is, $\vec{\alpha}$ is locally deterministic if (1) each tile added in $\vec{\alpha}$ "just barely" binds to the assembly; (2) if a tile of type t_0 at a location $\vec{m}$ and its immediate "OUT-neighbors" are deleted from the *result* of $\vec{\alpha}$, then no tile of type $t \neq t_0$ can attach itself to the thus-obtained configuration at location $\vec{m}$; and (3) the result of $\vec{\alpha}$ is terminal.

Definition. A GTAS $\mathcal{T} = (T, \sigma, \tau)$ is *locally deterministic* if there exists a locally deterministic τ-T-assembly sequence $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ with $\alpha_0 = \sigma$.

Theorem 3. (Soloveichik and Winfree [54]) Every locally deterministic GTAS is directed.

3.3 Pseudoseeds and Multiseeds

This section introduces two conceptual tools that enable us to reason about tile assembly systems in a modular fashion.

The idea of our first tool is intuitive. Suppose that our objective is to design a tile assembly system $\mathcal{T}$ in which a given set $X \subseteq \mathbb{Z}^2$ self-assembles. The set X might have a subset X^* for which it is natural to decompose the design into the following two stages.

(i) Design a TAS $\mathcal{T}_0 = (T_0, \sigma, \tau)$ in which an assembly σ^* with domain X^* self-assembles.

(ii) Extend T_0 to a tile set T such that X self-assembles in the TAS $\mathcal{T}^* = (T, \sigma^*, \tau)$

We would then like to conclude that X self-assembles in the TAS $\mathcal{T} = (T, \sigma, \tau)$. This will not hold in general, but it does hold if (i) continues to hold with T in place of T_0 and σ^* is a pseudoseed in the following sense.

www.manaraa.com

Definition. Let $\mathcal{T} = (T, \sigma, \tau)$ be a GTAS. A *pseudoseed* of $\mathcal{T}$ is an assembly $\sigma^* \in \mathcal{A}[\mathcal{T}]$ with the property that, if we let $\mathcal{T}^* = (T, \sigma^*, \tau)$, then, for every assembly $\alpha \in \mathcal{A}[\mathcal{T}]$, there exists an assembly $\alpha' \in \mathcal{A}[\mathcal{T}^*]$ such that $\alpha \sqsubseteq \alpha'$.

The following lemma says that the above definition has the desired effect.

Lemma 4 (pseudoseed lemma)**.** If σ^* is a pseudoseed of a GTAS $\mathcal{T} = (T, \sigma, \tau)$ and $\mathcal{T}^* = (T, \sigma^*, \tau)$, then $\mathcal{A}_\square[\mathcal{T}] = \mathcal{A}_\square[\mathcal{T}^*]$.

Proof. ("$\subseteq$" direction) Let $\alpha \in \mathcal{A}_\square[\mathcal{T}]$. This means that $\sigma \to \alpha$. The definition of pseudoseed tells us that there exists $\alpha' \in \mathcal{A}[\mathcal{T}^*]$ such that $\alpha \sqsubseteq \alpha'$. But since α is terminal, we must have $\alpha = \alpha'$, whence $\sigma \sqsubseteq \sigma^* \sqsubseteq \alpha$. It follows, by Rothemund's lemma (Lemma 2 on p. 57 of [46]), that $\sigma^* \to \alpha$.

("$\supseteq$" direction) Let $\alpha \in \mathcal{A}_\square[\mathcal{T}^*]$. Then we have $\sigma^* \to \alpha$. Moreover, $\sigma \to \sigma^*$ because $\sigma^* \in \mathcal{A}[\mathcal{T}]$. Thus, $\sigma \to \sigma^* \to \alpha$, and it follows, by the transitivity of $\to$ for single-temperature tile assembly systems, that $\alpha \in \mathcal{A}_\square[\mathcal{T}]$. □

Note that the pseudoseed lemma entitles us to *reason as though* the self-assembly proceeds in stages, even though this may not actually occur. (e.g., the pseudoseed σ^* may itself be infinite, in which case the self-assembly *of* σ^* and the self-assembly *from* σ^* must occur concurrently.)

Our second tool for modular reasoning is a bit more involved. Suppose that we have a tile set T and list $\sigma_0, \sigma_1, \sigma_2, \ldots$ of seeds that, for each i, the TAS $\mathcal{T}_i = (T, \sigma_i, \tau)$ has a desired assembly α_i as its result. If the assemblies $\alpha_0, \alpha_1, \alpha_2, \ldots$ have disjoint domains, then it might be possible for *all* these assemblies to grow from a "multiseed" σ^* that has $\sigma_0, \sigma_1, \sigma_2, \ldots$ embedded in it. We now define a sufficient condition for this.

Definition. Let T and T' be sets of tile types with $T \subseteq T'$, and let $\vec{\sigma} = (\sigma_i \mid 0 \leq i < k)$ be a sequence of τ-T-assemblies, where $k \in \mathbb{Z}^+ \cup \{\infty\}$. A *$\vec{\sigma}$-$T$-$T'$-multiseed* is a τ-T' assembly σ^* with the property that, if we write

$$\mathcal{T}^* = (T', \sigma^*, \tau)$$

www.manaraa.com

and

$$\mathcal{T}_i = (T, \sigma_i, \tau)$$

for each $0 \leq i < k$, then the following four conditions hold.

1. For each i, $\sigma_i \sqsubseteq \sigma^*$.
2. For each $i \neq j$, $\alpha \in \mathcal{A}[\mathcal{T}_i]$, $\alpha' \in \mathcal{A}[\mathcal{T}_j]$, $\vec{m} \in \text{dom } \alpha$, and $\vec{m'} \in \text{dom } \alpha'$, $\vec{m} - \vec{m'} \in U_2 \cup \{\vec{0}\} \Rightarrow \vec{m}, \vec{m'} \in \text{dom } \sigma^*$. (Recall that U_2 is the set of unit vectors in $\mathbb{Z}^2$.)
3. For each i, and each $\alpha \in \mathcal{A}[\mathcal{T}_i]$, there exists $\alpha^* \in \mathcal{A}[\mathcal{T}^*]$ such that $\alpha \sqsubseteq \alpha^*$.
4. For each $\alpha^* \in \mathcal{A}[\mathcal{T}^*]$, there exists, for each $0 \leq i < k$, $\alpha_i \in \mathcal{A}[\mathcal{T}_i]$ such that $\alpha^* \sqsubseteq \sigma^* + \sum_{0 \leq i < k} \alpha_i$.
5. For each $\alpha \in \mathcal{A}[\mathcal{T}^*]$, $\alpha^{-1}(T' - T) \subseteq \text{dom } \sigma^*$.

Note: In condition 4 we are using the operation $+$ defined as follows. If $\alpha, \alpha' : \mathbb{Z}^2 \dashrightarrow T$ are *consistent*, in the sense that they agree on $\text{dom } \alpha \cap \text{dom } \alpha'$, then $\alpha + \alpha' : \mathbb{Z}^2 \dashrightarrow T$ is the unique partial function satisfying $\text{dom } (\alpha + \alpha') = \text{dom } \alpha \cup \text{dom } \alpha'$, $\alpha \sqsubseteq \alpha + \alpha'$, and $\alpha' \sqsubseteq \alpha + \alpha'$. This is extended to summations $\sum_{0 \leq i < k} \alpha_i$ in the obvious way. The assemblies being summed in condition 4 are consistent by conditions 1 and 2.

Intuitively, the four conditions in the above definition can be stated as follows.

1. The seeds σ_i are embedded in σ^*.
2. Assemblies in $\mathcal{A}[\mathcal{T}_i]$ and assemblies in $\mathcal{A}[\mathcal{T}_j]$ do not interfere with each other.
3. σ^* does not interfere with assemblies in $\mathcal{A}[\mathcal{T}_i]$.
4. σ^* does not produce anything other than what its embedded seeds σ_i produce.
5. Tile types in $T' - T$ cannot occur outside σ^*.

The following lemma says that the multiseed definition has the desired effect.

www.manaraa.com

Lemma 5 (multiseed lemma)**.** Let $T \subseteq T'$ be sets of tile types, and let $\vec{\sigma} = (\sigma_i \mid 0 \leq i < k)$ be a sequence of τ-T-assemblies, where $k \in \mathbb{Z}^+ \cup \{\infty\}$. If σ^* is a $\vec{\sigma}$-T-T'-multiseed of $\mathcal{T}^*$ and $\mathcal{T}_i$ $(0 \leq i < k)$ are defined as in the multiseed definition, then

$$\mathcal{A}_{\square}[\mathcal{T}^*] = \left\{ \sigma^* + \sum_{0 \leq i < k} \alpha_i \;\middle|\; \text{each } \alpha_i \in \mathcal{A}_{\square}[\mathcal{T}_i] \right\}.$$

Proof. ("$\subseteq$" direction) Let $\alpha^* \in \mathcal{A}[\mathcal{T}^*]$. It follows by part (4) of definition 3.3 that $\alpha^* = \sigma^* + \sum_{0 \leq i < k} \alpha_i$, where $\alpha_i \in \mathcal{A}[\mathcal{T}_i]$, Since α^* is terminal, for each i, $\alpha_i \in \mathcal{A}_{\square}[\mathcal{T}_i]$, whence

$$\alpha^* \in \left\{ \sigma^* + \sum_{0 \leq i < k} \alpha_i \;\middle|\; \text{each } \alpha_i \in \mathcal{A}_{\square}[\mathcal{T}_i] \right\}$$

.

("$\supseteq$" direction) For each $0 \leq i < k$, let $\alpha_i \in \mathcal{A}_{\square}[\mathcal{T}_i]$ and let $\alpha = \sigma^* + \sum_{0 \leq i < k} \alpha_i$. It suffices to shaw that $\alpha \in \mathcal{A}_{\square}[\mathcal{T}^*]$.

Condition 3 of the multiseed definition tells us that, for each $0 \leq i < k$, there is an assembly $\alpha_i^* \in \mathcal{A}[\mathcal{T}^*]$ such that $\alpha_i \sqsubseteq \alpha_i^*$. Since $\sigma^* \sqsubseteq \alpha_i^*$, it follows that

$$\sigma^* + \alpha_i \sqsubseteq \alpha_i^*.$$

By condition 4 of the multiseed definition, there is, for each $0 \leq i < k$ and $0 \leq j < k$, an assembly $\alpha_{ij} \in \mathcal{A}[\mathcal{T}_i]$ such that, for all $0 \leq i < k$,

$$\alpha_i^* \sqsubseteq \sigma^* + \sum_{0 \leq j < k} \alpha_{ij}.$$

For each $0 \leq i < k$, let

$$D_i = \text{dom } \alpha_i^* \cap (\text{dom } \sigma^* \cup \text{dom } \alpha_{ii}),$$

and let

$$\hat{\alpha}_i^* = \alpha_i^* \upharpoonright D_i.$$

By (1), (2), and condition 2 of the multiseed definition, we have

$$\sigma^* + \alpha_i \sqsubseteq \hat{\alpha}_i^*$$

www.manaraa.com

for all $0 \leq i < k$. Now the assemblies $\hat{\alpha}_i^*$ are all consistent with one another, and each α_i is terminal in $\mathcal{A}[\mathcal{T}_i]$, so, by condition 5 of the multiseed definition, we must have

$$\alpha = \sum_{0 \leq i < k} \hat{\alpha}_i^*.$$

For each $0 \leq i < k$, let $\vec{\alpha}_i^*$ be an assembly sequence from σ^* to α_i^*, and let $\vec{\hat{\alpha}}_i^*$ be the sequence obtained from $\vec{\alpha}_i^*$ by deleting all additions of tiles not in $\sigma^* + \alpha_{ii}$. By condition 2 of the multiseed definition, each $\vec{\hat{\alpha}}_i^*$ is a τ-T'-assembly sequence from σ^* to $\hat{\alpha}_i^*$. By (3), then, if we dovetail the assembly sequences $\vec{\hat{\alpha}}_i^* (0 \leq i < k)$, we get an assembly sequence $\vec{\hat{\alpha}}^*$ from σ^* to α, whence $\alpha \in \mathcal{A}[\mathcal{T}^*]$. Since the α_i are terminal, conditions 4 and 5 of the multiseed definition tell us that $\alpha \in \mathcal{A}_{\square}[\mathcal{T}^*]$. □

3.4 Self-Assembly of Computably Enumerable Sets

In [58], Winfree proved that the Tile Assembly Model is Turing universal in two dimensions. In this section, we prove the stronger result: for every TM M, there exists a TAS that uniquely simulates M on (the binary representation of) *every* input $x \in \mathbb{N}$ in the two dimensional discrete Euclidean plane. We state our result precisely in the following theorem.

Theorem 6. Let $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ satisfying

$$f(n) = \binom{n+1}{2} + (n+1)\lfloor \log n \rfloor + 6n - 2^{1+\lfloor \log(n) \rfloor} + 2.$$

For all $A \subseteq \mathbb{Z}^+$, A is computably enumerable if and only if the set $X_A = \{(f(n), 0) \mid n \in A\}$ weakly self-assembles.

"⇐" direction. Assume the hypothesis. Then there exists a 2-TAS $\mathcal{T} = (T, \sigma, \tau)$ in which the set X_A self-assembles. Let B be the set of "black" tile types given by the definition of self-assembly. Fix some enumeration $\vec{a}_1, \vec{a}_2, \vec{a}_3 \ldots$ of $\mathbb{Z}^2$, and let M be the TM, defined as follows.

www.manaraa.com

Require: $n \in \mathbb{N}$

$\alpha := \sigma$

while $(f(n), 0) \notin \text{dom } \alpha$ **do**

choose the least $j \in \mathbb{N}$ such that $\vec{a}_j \in \partial^\tau \alpha_i$

choose $t \in T$ such that $\vec{a}_j \in \partial_t^\tau \alpha_i$

$\alpha := \alpha + (\vec{a}_j \mapsto t)$

end while

if $\alpha((f(n), 0)) \in B$ **then**

accept

else

reject

end if

It is clear from above that $L(M) = A$, whence A is computably enumerable. □

To prove the "⇒" direction, we exhibit, for any TM M, a directed TAS $\mathcal{T}_M = (T, \sigma, \tau)$ that correctly simulates M on all inputs $x \in \mathbb{Z}^+$ in the two dimensional discrete Euclidean plane. We describe our construction, and give the complete specification for T in the remainder of this section.

3.4.1 Overview of Construction

Intuitively, $\mathcal{T}_M$ self-assembles a "gradually thickening bar", immediately below the positive x-axis with upward growths emanating from well-defined intervals of points. For each $x \in \mathbb{Z}^+$, there is an upward growth that simulates M on x. If M halts on x, then (a portion of) the upward growth associated with the simulation of $M(x)$ eventually stops, and sends a signal down along the right side of the upward growth via a one-tile-wide-path of tiles to the point $(f(x), 0)$, where a black tile is placed. See Figure 3.1 for a finite, yet intuitive snapshot of this infinite process.

Our tile assembly system $\mathcal{T}_M$ is divided into three modules: the ray, the planter, and the TM module, which control the spacing between successive simulations, the initiation of upward

www.manaraa.com

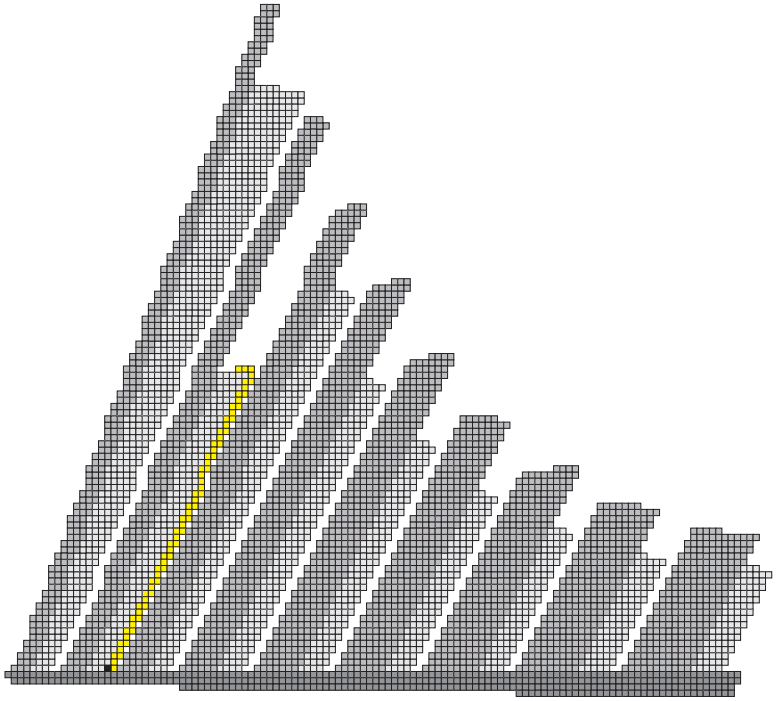


Figure 3.1: Simulation of M on every input $x \in \mathbb{N}$. Notice that $M(2)$ halts - indicated by the black tile along the x-axis.

www.manaraa.com

growth, and the actual simulations of M on each positive integer, respectively.

3.4.2 Overview of the Ray Module

The first module in our construction is the ray module (middle shade of gray squares in Figure 3.1). For any $3 \leq w \in \mathbb{Z}^+$, a ray of width w is a fixed-width, periodic, binary counter that repeatedly counts from 0 to $2^w - 1$, such that each integer is counted once, and then immediately copied once before the value of the binary counter is incremented. Essentially, a ray of width w is a discrete line of constant thickness w, having a kind of "slope" that depends on w in the following way. In every other row (except for two special cases), the first tile to attach does so on top of the second-to-leftmost tile in the previous row. Thus, a ray of width w will have a slope of $\frac{2^w}{2^{w-1}-1} = 2 + \frac{2}{2^{w-1}-1}$. This implies that the set of points occupied by properly spaced, consecutive rays of strictly increasing width, will not only be disjoint but the width of the gap in between such rays will increase without limit.

3.4.3 Details of Construction of the Ray Module

(It is to be understood that, although not explicit in our discussion, tile types in each module are colored with a "module indicator" symbol to prevent erroneous binding between the tile types of different modules.)

The ray module is a tile set of 68 tile types. Although in our construction the ray is not a self-contained tile assembly system, it can be made to be one with a trivial change. For any $3 \leq w \in \mathbb{N}$, a ray of width w is a fixed-width, periodic, binary counter that repeatedly counts from 0 to $2^w - 1$, such that each integer is counted once, and then immediately copied once before the value of the binary counter is incremented. Our ray is based on the binary counter from [47], and thus increments right-to-left, and then copies the previous value left-to-right. However, since we construct the ray to operate on the reverse of each integer (i.e., the LSB of each integer is its left most bit), increments proceed left-to-right and copies right-to-left. The most important feature of the ray is that the first, and hence leftmost, tile to attach in any increment row does so on top of the second-to-leftmost tile in the previous copy row, *unless*

www.manaraa.com

the bit pattern of the previous increment row is of the form $1^{w-1}(0+1)$. A detailed example of the former case is shown in Figure 3.2.

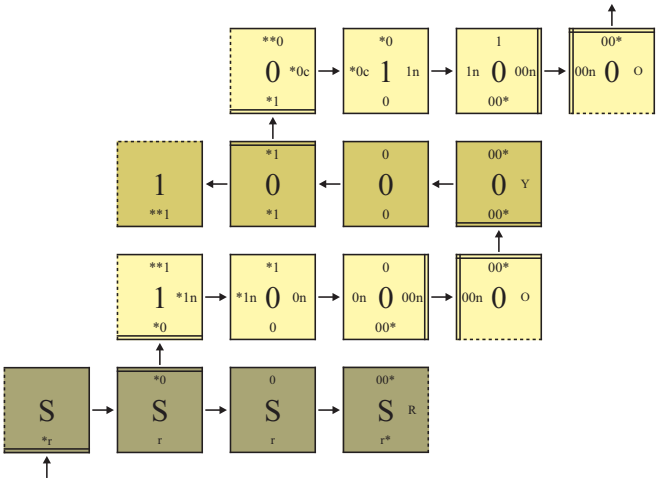


Figure 3.2: The first row of tiles is the initial row in a ray of width 4. The second row of tiles represents the value 1, and is the first increment row. The third row of tiles simply copies the value of the previous increment row.

In the latter case, the ray simply proceeds without "shifting" to the right by one unit. Copy rows are able to search for the two special bit patterns by using 'a' and 's' signals, respectively. For instance, when a copy row begins to self-assemble, and the current value of the counter is less than 2^{w-1}, an 'a' signal is propagated left through the copy row until a 0 bit is encountered in the previous increment row, thus breaking the signal. If no such bit exists, the signal will reach the leftmost tile of the copy row, and cause the next increment row to attach without being shifted one unit to the right. This situation is shown with detail in Figure 3.3.

Otherwise, the next increment row attaches on top of the second-to-leftmost tile in the current copy row. The 's' signal searches for the bit pattern 1^w in a similar fashion, shown with detail in Figure 3.4.

Finally, each row in the ray exhibits a particular "color" on the right side of its rightmost tile so as to allow a third (soon to be discussed) module to "know what to do and when to do it." Each increment row signals orange unless it contains the bit pattern $0^{w-1}(0+1)$, in which case it signals green for $0^{w-1}1$, and indigo otherwise. Each copy row signals yellow. However,

www.manaraa.com

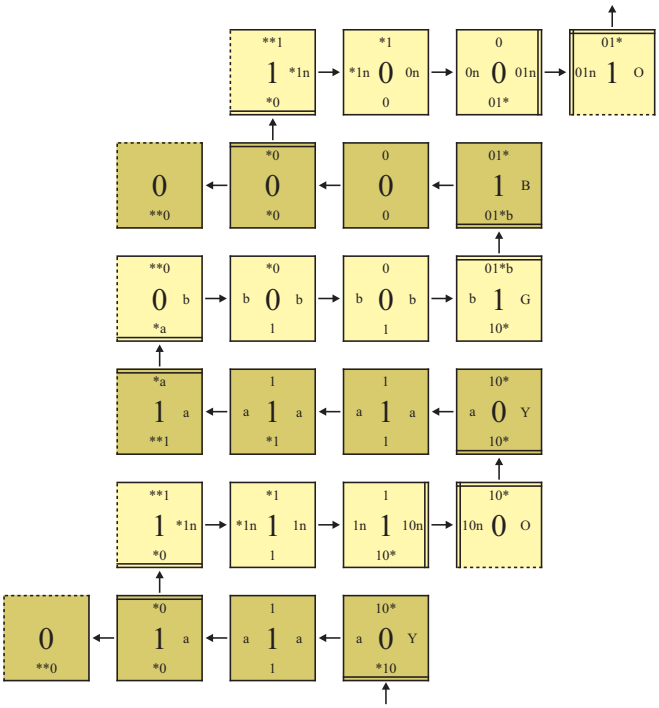


Figure 3.3: If the current value of the binary counter is a power of 2 (the fourth row of tiles), then the increment row that represents this value does not shift to the right by one unit. This situation is detected in the third (copy) row by the 'a' signal, which travels unimpeded right-to-left. The complementary 'b' signal is then sent left-to-right in the subsequent increment row to initiate a green signal.

if it is the copy row after an increment row with the bit pattern $0^{w-1}1$, then it signals blue. The initial row signals red.

In our construction, the upward growth of every ray is initiated by the planter. This can be seen with detail in Figure 3.5, where the top row of tiles (excluding the leftmost tile) will initiate the self-assembly of a ray having a width of 6. It is clear that our construction has the following properties.

1. A ray having a width of w will have a "slope" of $\frac{2^w}{2^{w-1}-1} = 2 + \frac{2}{2^{w-1}-1}$, which clearly tends to 2 as $w \to \infty$;

2. for every $3 \leq w \in \mathbb{Z}^+$, there is one and only one ray having a width of w;

3. thinner rays appear before thicker rays;

www.manaraa.com

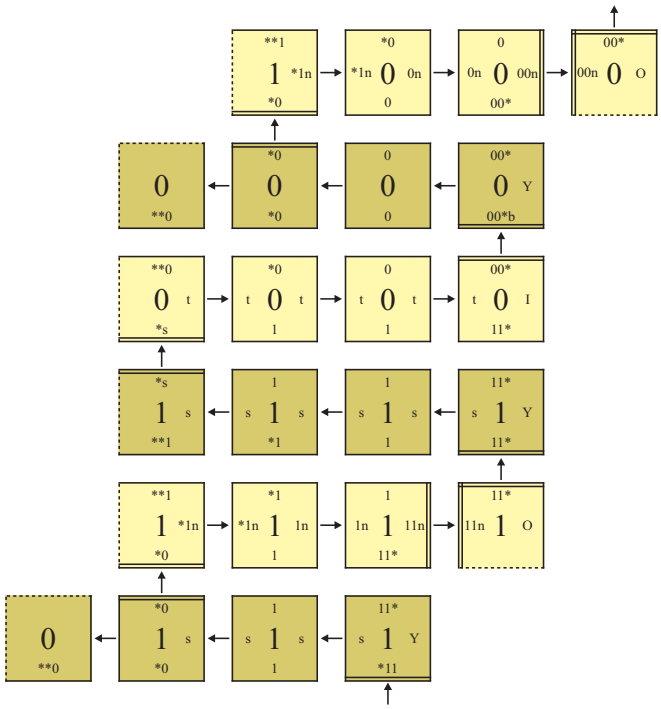


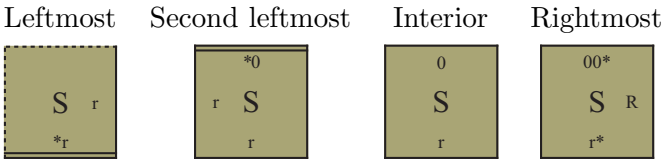
Figure 3.4: If the current value of the binary counter rolls over to all zeros (the fourth row of tiles), then the increment row that represents this value does not shift to the right by one unit. Similar to how the 'a' signal detects powers of 2, the 's' signal (the third row of tiles) detects when the counter is about to roll over. The 't' signal initiates an indigo signal.

4. the set of points occupied by any ray is disjoint from the set of points occupied by any other ray and

5. the width between successive rays grows without limit.

The following is a list of tile types that make up the ray module.

1. Initial row - add the following tile types:

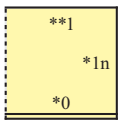


2. Tile types that perform general case increments (left to right).

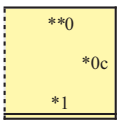
   (a) Leftmost tile - add the following tile types:

www.manaraa.com

No carry



Carry

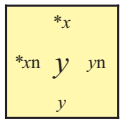


(b) Second leftmost tile type.

i. If there is no carry bit to propagate, then for all $x, y \in \{0, 1\}$, add the following tile types:

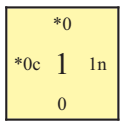
No carry in

No carry out

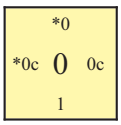


ii. If there is a carry bit coming in from the left, add the following tile types:

Carry in,
No carry out



Carry in,
Carry out

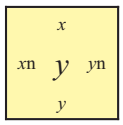


(c) Interior tile type.

i. If there is no carry bit to propagate, then for all $x, y \in \{0, 1\}$, add the following tile types:

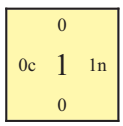
No carry in

No carry out

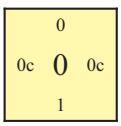


ii. If there is a carry bit coming in from the left, add the following tile types:

Carry in,
No carry out



Carry in,
Carry out



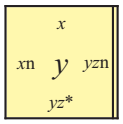
(d) Second rightmost tile type.

www.manaraa.com

i. If there is no carry bit to propagate, then for all $x, y, z \in \{0, 1\}$, add the following tile types:

No carry in

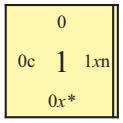
No carry out



ii. If the carry bit being propagated stops at the second most significant bit, then for all $x \in \{0, 1\}$, add the following tile types:

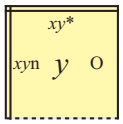
Carry in

No carry out



(e) Rightmost tile type - for all $x, y \in \{0, 1\}$, add the following tile type:

No carry in

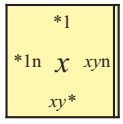


(f) Second right *and* second leftmost tile type (when $w = 3$.)

i. If there is no carry bit to propagate, then for all $x, y \in \{0, 1\}$, add the following tile types:

No carry in,

No carry out

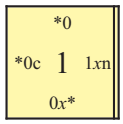


ii. If the carry bit stops at this location, then for all $x \in \{0, 1\}$, add the following tile types:

www.manaraa.com

Carry in,

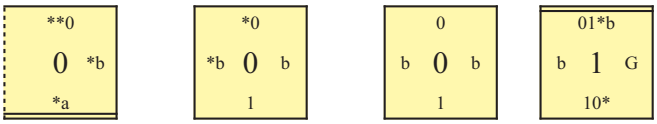
No carry out



3. Tile types that perform special case increments.

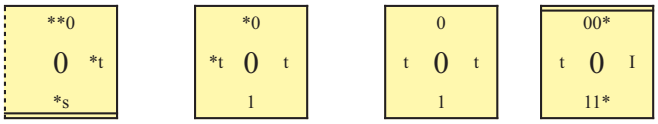
   (a) When the current row represents a power of 2 - add the following tile types:

Leftmost | Second leftmost | Interior | Rightmost



   (b) When the current row represents 0 - add the following tile types:

Leftmost | Second leftmost | Interior | Rightmost



4. Tile types that copy the current value of the counter (right to left).

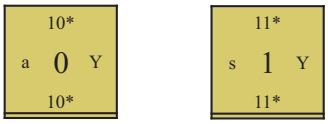
   (a) Rightmost tile type of copy row.

      i. Rightmost tile types that initiate a generic copy row - for all $y \in \{0, 1\}$, add the following tile types:

Search for bit pattern: $(0+1)^+0(0+1)$ | Copy row above bit pattern: 00^+1

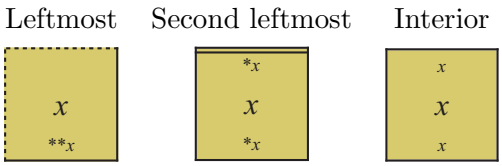
      ii. Rightmost tile types that search for a particular bit pattern - add the following tile types:

Search for bit pattern: 11^+0 | Search for bit pattern: 11^+

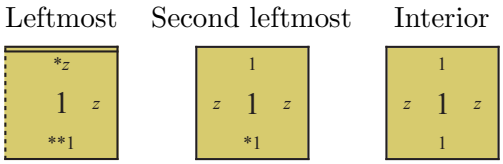


www.manaraa.com

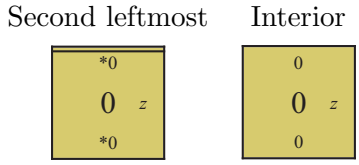
(b) Copy row interior tile types - for all $x \in \{0, 1\}$, add the following tile types:



(c) Copy row tile types that perform bit pattern search - for all $z \in \{\mathrm{a}, \mathrm{s}\}$ add the following tile types:



(d) Tile types that terminate bit pattern search - for all $z \in \{\mathrm{a}, \mathrm{s}\}$ add the following tile types:



3.4.4 Overview of the Planter Module

The next module is the planter module because it "plants the seeds" from which the ray modules will ultimately grow (the darkest gray squares in Figure 3.1). At the core of the planter module is a log-width, horizontal binary counter that counts every positive integer, starting at 1, in order. A key feature of the binary counter embedded in the planter module is that, after each integer is counted, a number of columns, equal to the current value of the binary counter, plus the number of bits in the binary representation of this value, plus a few extra "dummy" spacing columns, self-assemble. This has the effect of spacing out successive ray modules according to the function f (given in the introduction).

3.4.5 Details of Construction of the Planter Module

The tile set for the planter consists of 94 tile types. We partition the tile types into four logical subgroups.

www.manaraa.com

The first of the four subgroups is a standard binary counter that counts from 1 to infinity with the LSB of each integer having a y-coordinate of -1. The binary counter is based on an infinite fixed-width version of the binary counter in [47].

Suppose that $x \in \mathbb{Z}^+$ be the current value of the binary counter. The second subgroup receives x as input, and then in each subsequent row, subtracts one from the value of the number in the previous row until reaching 0, at which time a final "dummy" row consisting of all zeros self-assembles. This results in the self-assembly of exactly $x + 1$ rows following the binary counter row that represents x. The tile types that perform subtraction are based on the optimal binary counter (see [15]). Note that while subtraction is taking place, the current value of the binary counter is "remembered" via the rightmost bits in the east/west edge labels so that the value can be input to the third subgroup. Figure 3.5 shows a detailed example of the subtraction process for $n = 4$.

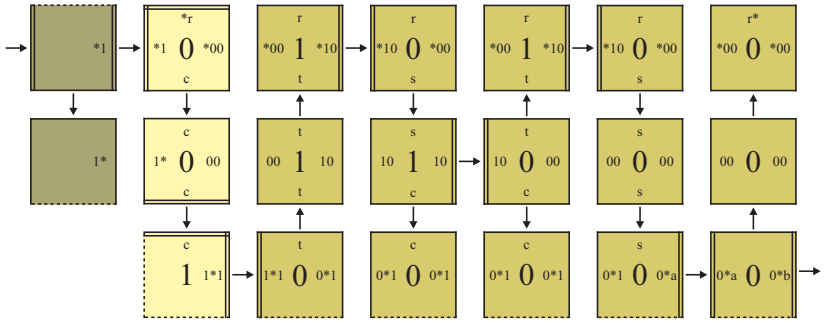


Figure 3.5: Subtraction in the planter. The first column of tiles represents the current value of the binary counter (4). The five rightmost columns of tiles perform subtraction.

Next, the third subgroup of tile types self-assembles into a square of size $\lfloor \lg x \rfloor + 1$ such that its north most edge labels are colored with the bits of x. The tile representing the LSB of x is the one that is placed in the upper right corner of the square. Notice that this has the affect of rotating and reflecting x up immediately below the x-axis. This is shown with detail in Figure 3.6. Notice that the rightmost tiles in Figure 3.5 bind with the leftmost tiles in Figure 3.6.

Finally, the fourth subgroup self-assembles three inert "dummy" spacing rows while allow-

www.manaraa.com

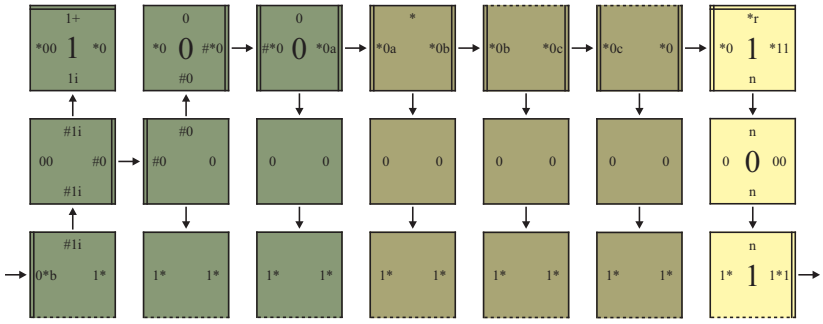
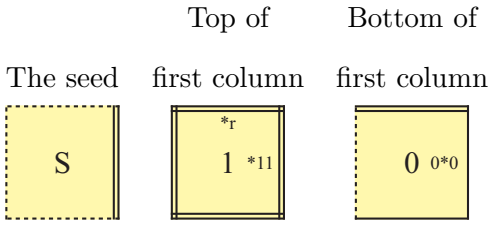


Figure 3.6: Rotation in the planter. The first three columns of tiles rotates and reflects the input (4) up immediately below the x-axis. The three subsequent columns of tiles are "dummy" spacing columns, and the final column of tiles represents the next value of the binary counter.

ing the current value of the binary counter to pass through. After these spacing rows attach, the next row of the binary counter self-assembles in which 1 is added to x. This process is repeated for all $x \in \mathbb{Z}^+$.

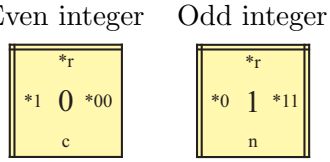
The following is the set of tile types that make up the planter module.

1. Seed tile types.



2. Binary counter tile types. The following tile types implement a fixed-width binary counter that counts every positive integer. Self-assembly proceeds naturally from LSB to MSB, which in our construction is top-to-bottom.

   (a) The first (topmost) tile type to attach (depending on whether the current row represents an even or odd integer) - add the following tile types:

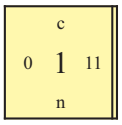


www.manaraa.com

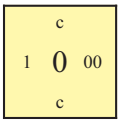
(b) Interior tile types that perform increments; The 'c' and 'n' characters symbolize situations in which there is either a carry bit coming in from the top or there is no carry bit.

i. If there is a carry bit coming in, add the following tile types:

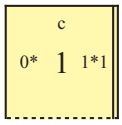
No carry bit out



Carry bit out

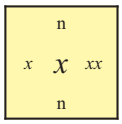


Bottom

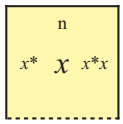


ii. If there is no carry bit coming in, then for all $x \in \{0, 1\}$, add the following tile types:

No carry bit out

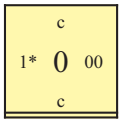


Bottom

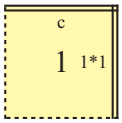


(c) Tile types that increase the number of bits in the binary counter - add the following tile types:

Propagate carry bit past border



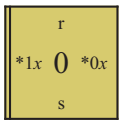
Increase width by 1 unit



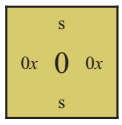
3. Subtraction tile types (a modification of the optimal binary counter given in [15]).

(a) Tile types that search for the least significant 1 bit in the current column - for all $x \in \{0, 1\}$, add the following tile types:

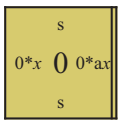
Top



Interior

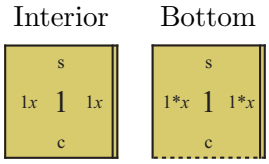


Bottom

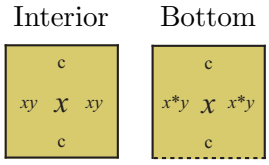


(b) Tile types that find the least significant 1 bit in the current column - for all $x \in \{0, 1\}$, add the following tile types:

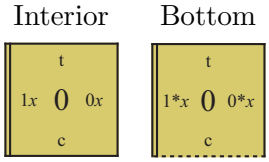
www.manaraa.com



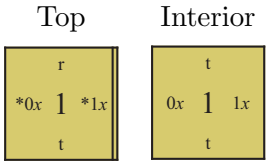
(c) Tile types that copy the least significant bits to the right (bottom) of the least significant 1 bit in the current column - for all $x, y \in \{0, 1\}$, add the following tile types:



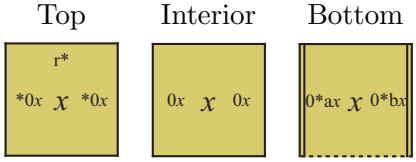
(d) The first tile type to attach in a row that represents an odd integer. Note that all tile types that attach above this tile type in a given column will represent the bit 1 (this is the purpose of the 't' signal) - for all $x \in \{0, 1\}$, add the following tile types:



(e) Tile types that convert all the least significant bits to the right(top) of the least significant 1 (in the previous column) to 1 - for all $x \in \{0, 1\}$, add the following tile types:



(f) After subtraction reaches 0, one final column is added - for all $x \in \{0, 1\}$, add the following tile types:



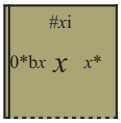
4. Rotation tile types.

www.manaraa.com

(a) Tile types of the initial column.

i. The first tile type - for all $x \in \{0, 1\}$, add the following tile types:

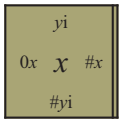
Bottom

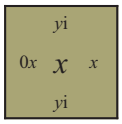


ii. Interior tile types - for all $x, y \in \{0, 1\}$, add the following tile types:

Second

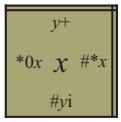
to attach In general

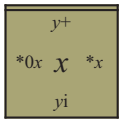




iii. Top most tile types - for all $x, y \in \{0, 1\}$, add the following tile types:

Special case In general



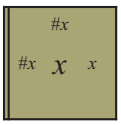


(b) Tile types that shift '#' up and to the right (not the initial or final column) while simultaneously copying the current value of the binary counter (the fourth subgroup) left-to-right, and bottom-to-top.

i. Shift right and then up - for all $x \in \{0, 1\}$, add the following tile types:

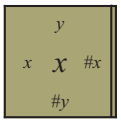
The first tile type

in a column



ii. Shift up and then right - for all $x, y \in \{0, 1\}$, add the following tile types:

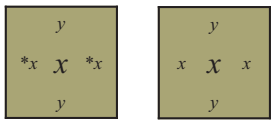
Topmost In general



iii. Tile types that do not participate in shifting the '#' token but are above the main diagonal - for all $x, y \in \{0, 1\}$, add the following tile types:

www.manaraa.com

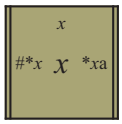
Topmost In general



(c) The first tile type to attach in the final column - for all $x \in \{0, 1\}$, add the following tile types:

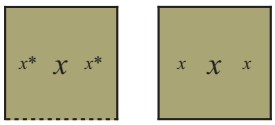
The first tile type

in a column



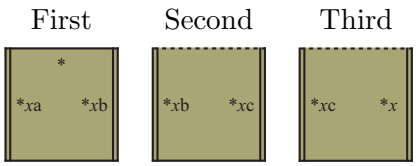
(d) Tile types that do not participate in shifting the '#' token but are below the main diagonal - for all $x \in \{0, 1\}$, add the following tile types:

Bottom In general



(e) Tile types that are responsible for the self-assembly of three inert spacing rows immediately after rotation - for all $x \in \{0, 1\}$, add the following tile types:

First Second Third



3.4.6 Overview of the Computation Module

The final module is, for any TM M, an algorithmically generated tile set that, in conjunction with the ray and planter modules, achieves the simulation of M on (the binary representation of) every input $x \in \mathbb{Z}^+$. The simulation of M on bin(x) proceeds vertically, immediately above the planter, while following the contour defined by the rightmost edge of the ray of width $x + 2$ (Note that by our construction of the planter module, there is one, and only one ray of such width). As with other standard Turing machine constructions (see [47, 54, 58]), each

www.manaraa.com

row in our simulation represents a configuration of M. However, the frequency with which transitions occur is a novel feature of our construction, and is controlled by "color" signals that are received from the abutting ray module.

3.4.7 Details of Construction of the Computation Module

Our simulation of M on x proceeds vertically while following the contour defined by the rightmost edge of the ray of width $x+2$. The right edge label of the rightmost tile type in every row of this ray sends a "color" signal that has the potential to initiate the self-assembly of a simulation row (if M halts on x, then eventually there will be no simulation rows to influence). The color signal defines the type of simulation row that subsequently self-assembles, whence self-assembly of every simulation row (except "blue" rows) proceeds left-to-right. In fact, all non-blue simulation rows, more or less, simply copy the previous configuration of M up one unit while, in some cases, also performing a shift-to-the-right by one unit.

Simulation begins with a row of tiles that represent the initial configuration of M with input x, which is represented by a row of $\lfloor \lg x \rfloor + 2$ red tile types (see below). The initial simulation row is adjacent to the rightmost tile of the initial row of the ray of width $x+2$, and immediately above the x-axis. All subsequent simulation rows either copy the configuration of M up to the next row (yellow and indigo rows) or do so while also shifting the contents to the right by one unit (orange rows). However, if a green color signal is received, then the simulation row increases the size of the working tape by a single tile (to the right), and then performs a single computation step in the next (blue) row. Note that when the size of the tape increases, the ray of width $x+2$ does not shift to the right, but the ray of width $x+3$ does. This relationship between successive rays maintains the constant width gap of 2 tiles between the simulation of M on x and the left border of the ray of width $x+3$.

If $M(x)\downarrow$, upward growth of the simulation halts, and a special signal is sent along a one-tile-wide path of tiles left-to-right along the top of the most recent simulation row. When the rightmost tile is encountered, the path continues down along the contour of the rightmost edge of the simulation until reaching the x-axis, at which point a black tile type is placed at

www.manaraa.com

the location $(f(x), 0)$. Since there is a constant gap (of width 2) between the right border of the simulation of M on x and the left border of the ray of width $x + 3$, the halting signal proceeds unimpeded.

The following is the set of tiles that make up the simulation module.

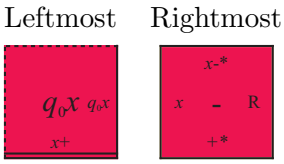
Let $M = (Q, \Sigma, \Gamma, \text{-}, \delta, q_0, q_h)$ be a Turing machine where

- Q is a finite set of states;
- $\Sigma = \{0, 1\}$ is the input alphabet;
- Γ is the tape alphabet;
- $\text{-} \in \Gamma - \Sigma$ is the blank symbol;
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$, is the transition function;
- $q_0 \in Q$ is the start state, and
- $q_h \in Q$ is the halting state.

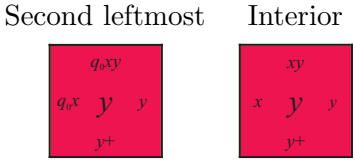
We make the assumption that M is initially in q_0 reading the leftmost symbol of its input $n \in \mathbb{N}$, which is in the leftmost cell on a one-way infinite tape. Further, we stipulate that M never moves left when reading the leftmost symbol on its tape.

1. Red (seed) tile types.

   (a) For all $x \in \Gamma$, add the following tile types:



   (b) For all $x, y \in \Gamma$, add the following tile types:

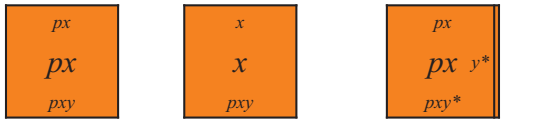


www.manaraa.com

2. Orange (copy) tile types.

   (a) If the location contains, or is next to the tape head, then for all $x, y \in \Gamma$, and for all $p \in Q$, add the following tile types:

   At tape head | Left of tape head | Left of rightmost (with tape head)

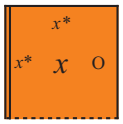
   (b) If the location is neither at or adjacent to the tape head, nor the rightmost location in the row, then for all $x, y \in \Gamma$, add the following tile types:

   Copy (no tape head) | Left of rightmost

   (c) Otherwise, for all $x \in \Gamma$, add the following tile types:

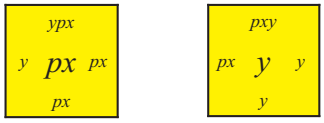
   Rightmost

3. Yellow tile types (shift tape contents right one unit).

   (a) For all $x, y \in \Gamma$, and for all $p \in Q$, add the following tile types:

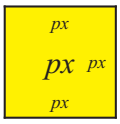
   Interior (with tape head) | Right of tape head

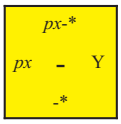
   (b) For all $x \in \Gamma$, and for all $p \in Q$, add the following tile types:

www.manaraa.com

Leftmost (with tape head)



Rightmost and adjacent to tape head

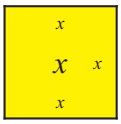


(c) For all $x, y \in \Gamma$, add the following tile types:

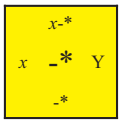
Interior

(d) For all $x \in \Gamma$, add the following tile types:

Leftmost (no tape head)



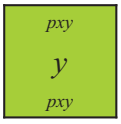
Rightmost



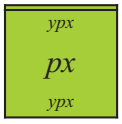
4. Green tile types (pre-transition):

(a) For all $x, y \in \Gamma$, and for all $p \in Q$, add the following tile types:

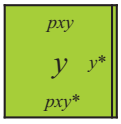
Interior (left of tape head)



Interior (with tape head)

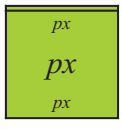


Left of rightmost (right of tape head)



(b) For all $x \in \Gamma$, and for all $p \in Q$, add the following tile types:

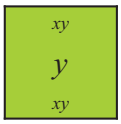
Leftmost (with tape head)



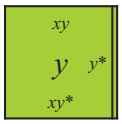
(c) For all $x, y \in \Gamma$, add the following tile types:

www.manaraa.com

Interior (no tape head)

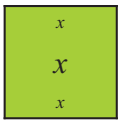


Left of rightmost

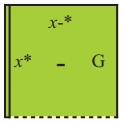


(d) For all $x \in \Gamma$, add the following tile types:

Leftmost (no tape head)



Rightmost

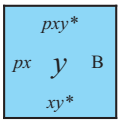


5. Blue tile types (Turing machine transition).

(a) Tile types that do not perform a transition.

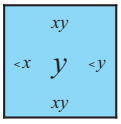
i. For all $x, y \in \Gamma$, and for all $p \in Q$, add the following tile types:

Rightmost and adjacent to tape head

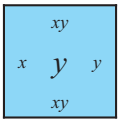


ii. For all $x, y \in \Gamma$, add the following tile types:

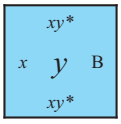
Interior (left of tape head)



Interior (right of tape head)



Rightmost and not adjacent to to tape head



iii. For all $x \in \Gamma$, add the following tile types:

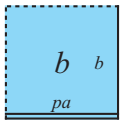
Leftmost

(b) Tile types that perform a right transition. If $\exists a, b \in \Gamma$ such that $(q, b, r) = \delta(p, a)$, then add the following tile types:

www.manaraa.com

Interior and

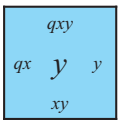
right of tape head



i. For all $x, y \in \Gamma$, add the following tile types:

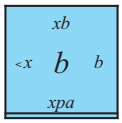
Interior and

right of tape head

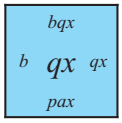


ii. For all $x \in \Gamma$, add the following tile types:

Start of transition at interior



Right of tape head after transition

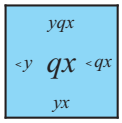


(c) Tile types that perform a left transition. If $\exists a, b \in \Gamma$ such that $(q, b, L) = \delta(p, a)$, then add the following tile types:

i. For all $x, y \in \Gamma$, add the following tile types:

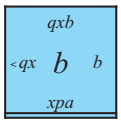
Interior and left

of tape head

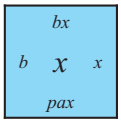


ii. For all $x \in \Gamma$, add the following tile types:

Start of transition at interior



Right of transition

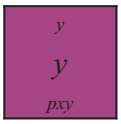


6. Indigo (copy tape contents straight up) tile types.

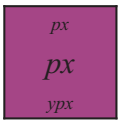
(a) For all $x, y \in \Gamma$, and for all $p \in Q$, add the following tile types:

www.manaraa.com

Interior (left of tape head)



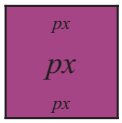
Rightmost, and next to tape head



Rightmost, and not next to tape head

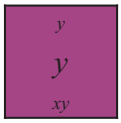
(b) For all $x \in \Gamma$, and for all $p \in Q$, add the following tile types:

Leftmost (with tape head)

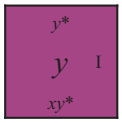


(c) For all $x, y \in \Gamma$, add the following tile types:

Interior without tape head

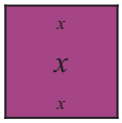


Rightmost



(d) For all $x \in \Gamma$, add the following tile types:

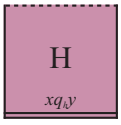
Leftmost (without tape head)



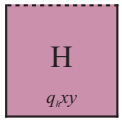
7. Violet (halting) tile types.

(a) For all $x, y \in \Gamma$, add the following tile types:

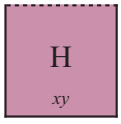
Initial interior halting tile type



Right of initial halting tile type



Fill in remainder of final row



(b) For all $x \in \Gamma$, add the following tile types:

www.manaraa.com

Initial halting tile type (leftmost)

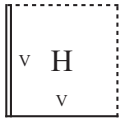
Leftmost tile type (no tape head)

Second rightmost tile type

8. Tile types that snake down the rightmost edge of a halting simulation.

   (a) The first tile type to attach:

Rightmost of halting row

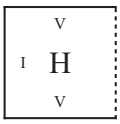


   (b) Grow down:

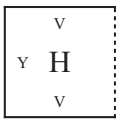
Green

Orange

Indigo

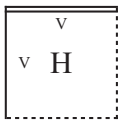


Yellow



   (c) Initiate left growth:

Grow left



   (d) Grow left one unit:

Bump into yellow row

Bump into blue row

9. If M halts on input n then the following tile type (the only "black" tile type in our construction) is placed at the point $(f(n), 0)$:

www.manaraa.com

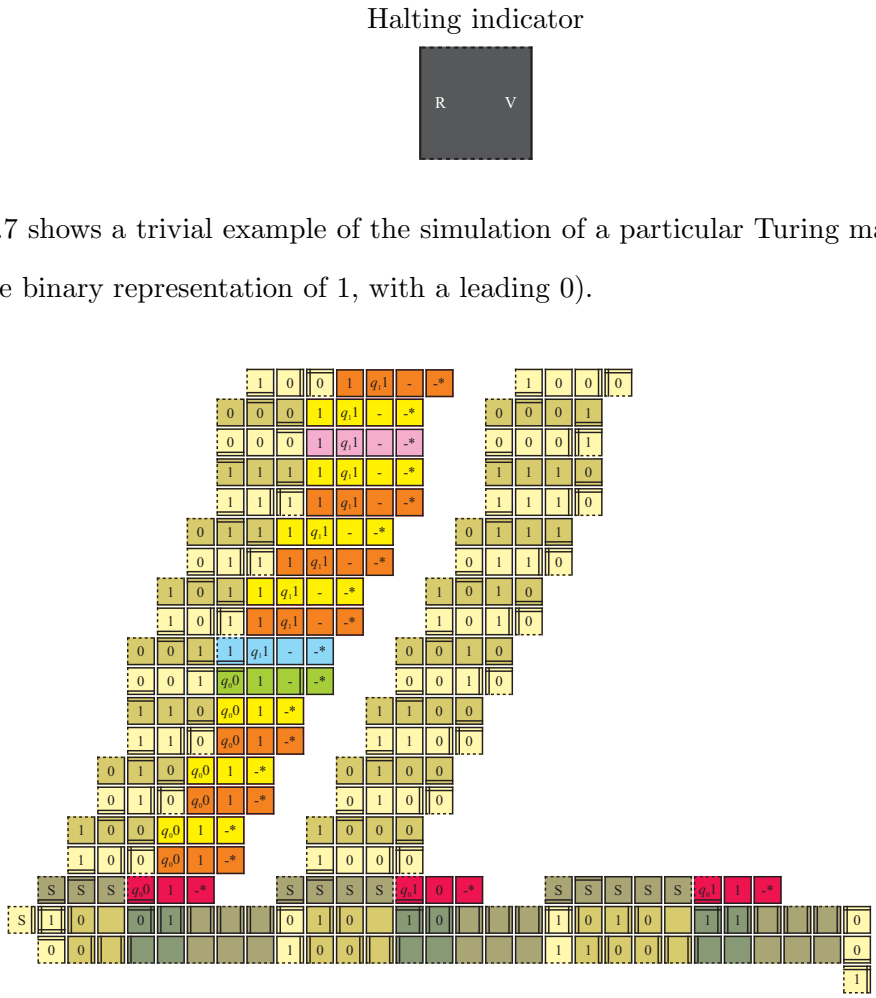


Figure 3.7 shows a trivial example of the simulation of a particular Turing machine M on input 01 (the binary representation of 1, with a leading 0).

Figure 3.7: A closer look at the simulation of M on input 1 (influenced by a ray of width 3). All three modules can be seen in this figure.

3.4.8 Proof of Correctness

Let $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ be as in section 1, stipulating that $f(0) = -1$. For each $n \in \mathbb{N}$, define the rectangle

$$Q_n = \{f(n-1) + 2, \ldots, f(n) - 1\} \times \{-2, -1\},$$

and define the following assemblies.

(i) σ_n is the portion of the planter lying in Q_n.

www.manaraa.com

(ii) ρ_n is the ray of width $n+2$, translated so that its base is the leftmost $2(n+2)$ tiles in σ_n.

(iii) $\sigma_n^* = \sigma_n + \rho_n$, where "+" is the operation defined in section 3.

(iv) γ_n is the assembly that simulates $M(n)$ as in section 4.4.

(v) σ^* is our planter.

(vi) $\alpha_n = \sigma_n^* + \gamma_n$.

(vii) $\alpha = \sigma^* + \sum_{n=0}^{\infty} \alpha_n$.

Let

$$\mathcal{T}_M = (T_M, \sigma, \tau)$$

be our tile assembly system, noting that σ consists of a single tile at the origin. Define the following subsets of $\mathcal{T}_M$.

(i) T_R is the set of tile types used in the ray module, together with the benign tile type, all of whose edges are given a binding strength of 1 with no color, appearing in the rectangles σ_n.

(ii) T_C is the set of tile types in T_R, together with those occurring in the computation module.

(iii) T_P is the set of tile types in the planter module.

For each $n \in \mathbb{Z}^+$, define the tile assembly systems

$$\mathcal{T}_{R,n} = (T_R, \sigma_n, \tau),$$
$$\mathcal{T}_{C,n} = (T_C, \sigma_n, \tau),$$
$$\mathcal{T}_{C,n}^* = (T_C, \sigma_n^*, \tau).$$

Lemma 7. For each $n \in \mathbb{Z}^+$, $\mathcal{A}_{\square}[\mathcal{T}_{R,n}] = \{\sigma_n^*\}$.

Proof. Define the infinite, τ-T_R-assembly sequence, $\vec{\alpha}$, to be that which is implicit in Figures 3.2, 3.3, and 3.4. It is easy to see from our construction that every tile that binds in $\vec{\alpha}$

www.manaraa.com

does so uniquely. Furthermore, every such tile addition occurs with exactly strength 2. It is clear that res($\vec{\alpha}$) is terminal, whence $\vec{\alpha}$ is a locally deterministic assembly sequence. □

Lemma 8. For each $n \in \mathbb{Z}^+$, σ_n^* is a pseudoseed of $\mathcal{T}_{C,n}$.

Proof. First, note that $\sigma_n^* \in \mathcal{A}[\mathcal{T}_{C,n}]$ because $\sigma_n^* \in \mathcal{A}[\mathcal{T}_{R,n}]$ and $T_R \subset T_C$.

Let $\alpha \in \mathcal{A}[\mathcal{T}_{C,n}]$. Let $\vec{\alpha}'$ be the τ-T_C-assembly sequence such that res($\vec{\alpha}'$) $= \alpha$. It is clear, since $\vec{\alpha}$ is locally deterministic and because of the use of module indicators in our construction, that when $\vec{\alpha}'$ assigns a tile type to a location in ρ_n, it does so in agreement with $\vec{\alpha}$ from Lemma 4. Thus, we can use $\vec{\alpha}$ to "extend" res($\vec{\alpha}'$) and thereby fill in the remainder of ρ_n. This gives us an assembly $\alpha' \in \mathcal{A}\left[\mathcal{T}_{C,n}^*\right]$ such that $\alpha \sqsubseteq \alpha'$. □

Lemma 9. For each $n \in \mathbb{Z}^+$, $\mathcal{A}_\square\left[\mathcal{T}_{C,n}^*\right] = \{\alpha_n\}$.

Proof. Define the τ-$T_{C,n}$-assembly sequence $\vec{\alpha}_n$ as that which self-assembles each simulation row one at a time, and according to the following rule: if the current row is not "blue," then, by our construction, self-assembly must proceed left to right; however, if the current row is "blue," then $\vec{\alpha}_n$ first attaches the initial tile, via a single τ-strength bond along its south edge, and then the remaining tiles in order of increasing distance from the origin. It is clear from our construction that every tile that binds in $\vec{\alpha}_n$ does so uniquely, and with exactly strength τ. Since res($\vec{\alpha}_n$) is terminal, it follows that $\vec{\alpha}_n$ is locally deterministic. □

Lemma 10. For each $n \in \mathbb{Z}^+$, $\mathcal{A}_\square[\mathcal{T}_{C,n}] = \{\alpha_n\}$.

Proof. This follows immediately from Lemma 4, Lemma 5, and the pseudoseed lemma. □

Define the tile assembly system

$$\mathcal{T}_P = (T_P, \sigma, \tau).$$

Lemma 11. $\mathcal{A}_\square[\mathcal{T}_P] = \{\sigma^*\}$.

Proof. Define the infinite τ-T_P-assembly sequence, $\vec{\alpha}$, to be that which self-assembles σ^* one row at a time, and implicit from Figures 3.5 and 3.6. Note that $\vec{\alpha}$ starts from the seed tile

www.manaraa.com

located at the origin. It is clear from our construction that every tile that binds via $\vec{\alpha}$ does so uniquely. Furthermore, every such tile addition occurs with exactly strength 2. It is clear that $\text{res}(\vec{\alpha})$ is terminal, whence $\vec{\alpha}$ is a locally deterministic assembly sequence. □

Lemma 12. σ^* is a pseudoseed of $\mathcal{T}_M$.

Proof. First, note that $\sigma^* \in \mathcal{A}[\mathcal{T}_M]$ because $\sigma^* \in \mathcal{A}[\mathcal{T}_P]$ and $T_P \subset T_M$.

Let $\alpha \in \mathcal{A}[\mathcal{T}_M]$. Let $\vec{\alpha}'$ be the τ-T_M-assembly sequence such that $\text{res}(\vec{\alpha}') = \alpha$. It is clear, since $\vec{\alpha}$ is locally deterministic and because of the use of module indicators in our construction, that when $\vec{\alpha}'$ assigns a tile to a location that is in σ^*, it does so in agreement with $\vec{\alpha}$ from Lemma 8. Thus, we can use $\vec{\alpha}$ to "extend" $\text{res}(\vec{\alpha}')$ and thereby fill in the remainder of σ^*. This gives us an assembly $\alpha' \in \mathcal{A}\left[\mathcal{T}^*_{C,n}\right]$ such that $\alpha \sqsubseteq \alpha'$. □

Lemma 13. σ^* is a $\vec{\sigma}$-T_C-T_M-multiseed.

Proof. Let $\vec{\sigma} = (\sigma_n \mid n \in \mathbb{Z}^+)$. By our construction of the planter, it is clear that part (1), of definition 3.3, is satisfied. To see that (2) is satisfied, we appeal to: Lemma 7, our construction of the ray, and the fact that the planter spaces out each σ_n sufficiently. Moreover, it is clear that (5) holds because of our use of module indicator symbols in our construction. We now turn our attention to parts (3) and (4).

Let $i \in \mathbb{Z}^+$, and $\alpha \in \mathcal{A}[\mathcal{T}_i]$. Let $\vec{\alpha}'$ be an assembly sequence such that $\alpha = \text{res}(\vec{\alpha}')$. Since (1) holds, we can define the assembly sequence $\vec{\alpha} = (\sigma^*, \ldots, \alpha)$, and let $\alpha^* = \text{res}(\vec{\alpha})$. It is clear that $\alpha^* \in \mathcal{A}[\mathcal{T}^*_M]$, and $\alpha \sqsubseteq \alpha^*$, whence (3) holds.

Finally, let $\alpha^* \in \mathcal{A}[\mathcal{T}_M]$. By (1), (2), Lemma 7, and our construction of the planter, it is easy to see that (4) holds.

□

Lemma 14. $\mathcal{A}_\square[\mathcal{T}_M] = \{\alpha\}$.

Proof. Let $\mathcal{T}^*_M = (T_M, \sigma^*, \tau)$. By Lemma 6, 7, and 9, and the multiseed lemma, $\mathcal{A}_\square[\mathcal{T}^*_M] = \{\alpha\}$. It follows by Lemma 8 and the pseudoseed lemma that $\mathcal{A}_\square[\mathcal{T}_M] = \{\alpha\}$. □

www.manaraa.com

3.5 A Decidable Set That Does Not Self-Assemble

We now show that there are decidable sets $D \subseteq \mathbb{Z}^2$ that do not weakly self-assemble in the Tile Assembly Model.

For each $r \in \mathbb{N}$, let

$$D_r = \{(m,n) \in \mathbb{Z}^2 \mid |m| + |n| = r\}.$$

This set is a "diamond" in $\mathbb{Z}^2$ (i.e., a circle in the taxicab metric) with radius r and center at the origin. For each $A \subseteq \mathbb{N}$, let

$$D_A = \bigcup_{r \in A} D_r.$$

This set is the "system of concentric diamonds" centered at the origin with radii in A. The following observation is used in the proof of Lemma 16.

Observation 15. $|D_{<r}| = \left|\{(m,n) \in \mathbb{Z}^2 \mid |m| + |n| < r\}\right| = 2(r-1)^2 + 2r - 1$.

Proof.

$$\begin{aligned} |D_{<r}| &= \left|\{(m,n) \in \mathbb{Z}^2 \mid |m| + |n| < r\}\right| \\ &= 2r - 1 + 2 \cdot \sum_{i=0}^{r-2} (2i+1) \\ &= 2r - 1 + 2(r-1)^2. \end{aligned}$$

□

Lemma 16. If $A \subseteq \mathbb{N}$ and D_A self-assembles, then $A \in \text{DTIME}\left(2^{4n}\right)$.

The proof of this lemma exploits the fact that a tile assembly system in which D_A self-assembles must, for sufficiently large r, decide the condition $r \in A$ from *inside* the diamond D_r.

Proof. If $|A| < \infty$ then we are done, so assume otherwise (i.e., A is infinite).

Assume the hypothesis. Then there exists a 2-TAS $\mathcal{T} = (T, \sigma, \tau)$ in which the set D_A self-assembles. Fix some enumeration $\vec{a}_1, \vec{a}_2, \vec{a}_3 \ldots$ of $\mathbb{Z}^2$, and let M be the TM, defined as follows, that simulates the self-assembly of $\mathcal{T}$.

www.manaraa.com

Require: $r \in \mathbb{N}$

$\alpha := \sigma$

while $D_r \cap \operatorname{dom} \alpha = \varnothing$ **do**

choose the least $j \in \mathbb{N}$ such that $\vec{a}_j \in \partial\alpha_i$

choose $t \in T$ such that $\vec{a}_j \in \partial_t \alpha_i$

$\alpha := \alpha + (\vec{a}_j \mapsto t)$

end while

if $\alpha(\vec{a}_j) \in B$ **then**

accept

else

reject

end if

Observe that the *while* loop will terminate after at most $|D_{<r}| + 1 - |\operatorname{dom} \alpha|$ iterations, with $D_r \cap \operatorname{dom} \alpha \neq \varnothing$ whence M halts on all inputs. If $r \in A$, then the weak self-assembly of D_A tells us that for every $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$, $\alpha(D_r) \subseteq B$. Since we have $\vec{a}_j \in D_r$ when the *while* loop terminates, M accepts, and $r \in L(M)$. If $r \in L(M)$, then $\alpha(\vec{a}_j) \in B$, whence $r \in A$.

By Observation 15, the *while* loop performs at most $2(r-1)^2 + 2r - 1$ iterations, and in each iteration, we are doing at most $O\left(r^2\right)$ amount of additional work checking $D_r \cap \operatorname{dom} \alpha = \varnothing$, and maintaining $\partial^\tau \alpha_i$. Thus, M decides A in $O\left(2^{4n}\right)$ computation steps, where $n = \lfloor \log r \rfloor + 1$ (i.e., the length of r).

Note: Implicit in this proof is the fact, proven by Irani, Naor, and Rubinfeld [**?**], that write-once memory is equivalent to time. □

We now have the following result.

Theorem 17. There is a decidable set $D \subseteq \mathbb{Z}^2$ that does not weakly self-assemble.

Proof. By the time hierarchy theorem [25], there is a set $A \subseteq \mathbb{N}$ such that

$$A \in \operatorname{DTIME}\left(2^{5n}\right) - \operatorname{DTIME}\left(2^{4n}\right).$$

Let $D = D_A$. Then D is decidable and, by Lemma 16, D does not self-assemble. □

www.manaraa.com

Note that both Lemma 16 and (hence) Theorem 18 hold for *any* value of $\tau > 0$.

3.6 Conclusion

The first theorem of this chapter established that, for every computably enumerable set $A \subseteq \mathbb{Z}^+$, the representation $X_A = \{(f(n), 0) \mid n \in A\}$ weakly self-assembles. This representation of A is somewhat sparse along the x-axis, because our f grows quadratically. A linear function f would give a more compact representation of A. We conjecture that our first main theorem does *not* hold for any linear function, but we do not know how to prove this.

Let D be the set presented in the proof of the second theorem in this chapter. It is easy to see that the condition $(m, n) \in D$ is decidable in time polynomial in $|m| + |n|$, but $|m| + |n|$ is exponential in the length of the binary representation of (m, n), so this only tells us that $D \in \mathrm{E} = \mathrm{DTIME}\left(2^{\mathrm{linear}}\right)$. Is there a set $D \subseteq \mathbb{Z}^2$ such that $D \in \mathrm{P}$, and D does not weakly self-assemble?

More generally, we hope that the results presented in this chapter lead to further research illuminating the interplay between geometry and computation in self-assembly.

www.manaraa.com

CHAPTER 4. Geometric Universality in the Multiple Temperature Model

4.1 Introduction

The aTAM been studied considerably from the perspective of computational complexity theory. A problem that has received substantial attention is that of finding "small" tile sets that assemble $N \times N$ squares in the aTAM. For instance, Adleman, Cheng, Goel, and Huang [2] proved that $N \times N$ squares self-assemble with $O\left(\frac{\log N}{\log \log N}\right)$ distinct tile types, matching the Kolmogorov-dictated lower bound that was established in [47]. The more general problem of the self-assembly of arbitrary shapes in the aTAM has also been considered. Most notably, Soloveichik and Winfree [54] discovered a beautiful connection between the the Kolmogorov complexity of an arbitrary scaled shape and the minimum number of tile types required to assemble it. This last result has the property that it is *geometrically* universal in the sense that any finite shape can be produced using a very simple tile set.

In addition to being an elegant and powerful theoretical tool, there is also experimental justification for the aTAM. For example, using DNA double-crossover molecules to construct tiles only a few nanometers long, Rothemund, Papadakis and Winfree [48] implemented the molecular self-assembly of the well-known fractal structure called the *discrete Sierpinski triangle* with low enough error rates to achieve correct placement of 100 to 200 tiles. Moreover, Barish, Schulman, Rothemund and Winfree [6] have recently used Rothemund's DNA origami [45] as a seed structure to which subsequent "computation" DNA tiles can attach and assemble computationally interesting patterns with error rates less than .2%! Note that this technique, although robust, is not general-purpose in the sense that all of the information about the to-be-assembled shape (or pattern) is encoded into the DNA tiles and origami seed.

In fact, a central problem in algorithmic self-assembly is that of *providing input to a tile*

www.manaraa.com

assembly system (e.g., the size of a square, the description of a shape, etc.). In real-world laboratory implementations, as well as theoretical constructions, input to a tile system in the aTAM is provided via a (possibly large) collection of "hard-coded" seed tile types [2, 6, 47, 54]. Unfortunately in practice, it is more expensive to manufacture many different types of tiles, as opposed to creating several copies of each tile type. This suggests that it might be advantageous to be able to provide input to a tile system without having to resort to hard-coding the input into a large number its own tiles. As a result, several natural generalizations of the aTAM have been developed in an attempt to model various types of alternative input delivery mechanisms.

One such model is the *staged self-assembly* model [16, 43], in which several intermediate structures are allowed to assemble in different test tubes before they are all mixed together to obtain the target structure. Demaine, Demaine, Fekete, Ishaque, Rafalin, Schweller, and Souvaine [16] proved that arbitrary shapes self-assemble with $O(1)$ tile types but with a corresponding increase in the number of stages and even (in some cases) an increase in the scale of the target shape. Note that, in the staged self-assembly model, the input to a tile system is implicitly encoded in the actions of the laboratory scientist and not in the tile types themselves.

Another means of providing input to a tile system is through the programming of the relative concentrations of its tile types. Becker, Rapaport, and Rémila [8] proved that by appropriately setting the relative concentrations of tiles, squares, rectangles and diamonds can self-assemble in an expected sense with $O(1)$ tile types, but with a large (and undesirable) variance. Kao and Schweller [27] improved the aforementioned result by showing that it is possible to program the relative concentrations of $O(1)$ tile types such that they will assemble into arbitrarily close approximations of $N \times N$ squares with high probability. Furthermore, Doty [17] recently showed that $N \times N$ squares self-assemble *exactly* with high probability with $O(1)$ tile types.

Input can also be delivered to a tile system through the deliberate variation of its temperature. The *multiple temperature* model [14, 26] is a natural generalization of the aTAM, where the temperature of a tile system is dynamically adjusted by the experimenter as self-assembly proceeds. Aggarwal, Cheng, Goldwasser, Kao, and Schweller [14] proved that the number of

www.manaraa.com

tile types required to assemble "thin" $k \times N$ rectangles can be reduced from $O\left(\frac{N^{1/k}}{k}\right)$ (in the aTAM) to $O\left(\frac{\log N}{\log \log N}\right)$ if the temperature is allowed to change but once. Subsequently, Kao and Schweller [26] discovered a clever "bit-flipping" scheme capable of assembling any $N \times N$ square using $O(1)$ tile types and $\Theta(\log N)$ temperature changes. Note that the multiple temperature model has a similar flavor to that of the staged self-assembly model in the sense that the input to a tile system in both models can be encoded into a sequence of laboratory operations.

In all of the results mentioned in the previous three paragraphs, with the notable exception of [16], attention was focused on the problem of reducing the number of distinct tile types needed for the assembly of certain *restricted* classes of shapes such as diamonds, thin rectangles or squares. In this chapter, we study the broader problem of reducing the number of tiles needed to assemble *arbitrary* finite shapes in the multiple temperature model.

In particular, we exhibit two constant-size tile sets in which scaled-up versions of arbitrary shapes self-assemble. Our first tile set has the property that each scaled shape self-assembles via a temperature sequence whose length is proportional to the Kolmogorov complexity of the shape, but the scaling factor grows with the size of the shape being assembled. In contrast, our second tile set assembles each scaled shape via a temperature sequence whose length is proportional to the number of points in the shape but the scaling factor is a constant independent of the shape being assembled. Both of these constructions are geometrically universal in the sense that they can be programmed to self-assemble into any finite shape via appropriate changes in the system temperature.

Finally, we show that the scale factor in both of our constructions is necessary, i.e., that there is no constant-size tile set that can uniquely assemble an arbitrary shape in the multiple temperature model. This last result answers an open question of Kao and Schweller [26], who asked whether such a tile system existed.

www.manaraa.com

4.2 Self-Assembly of Arbitrary Scaled Shapes with $O(1)$ Tile Types

In this section, we exhibit two constructions that are capable of building scaled-up versions of arbitrary shapes in the multiple temperature model. Fix some universal Turing machine U. The *Kolmogorov complexity* of a shape X, denoted as $K(X)$, is the size of the smallest program π that outputs an encoding of a list of all the points in X. In other words $K(X) = \min\{|\pi| \mid U(\pi) = \langle S \rangle\}$. The reader is encouraged to consult [33] for a more detailed discussion of Kolmogorov complexity.

Both constructions reduce the tile complexity for the self-assembly of arbitrary scaled shapes from $\Theta\left(\frac{K(X)}{\log K(X)}\right)$ [54] to $O(1)$, but with a corresponding increase in temperature complexity.

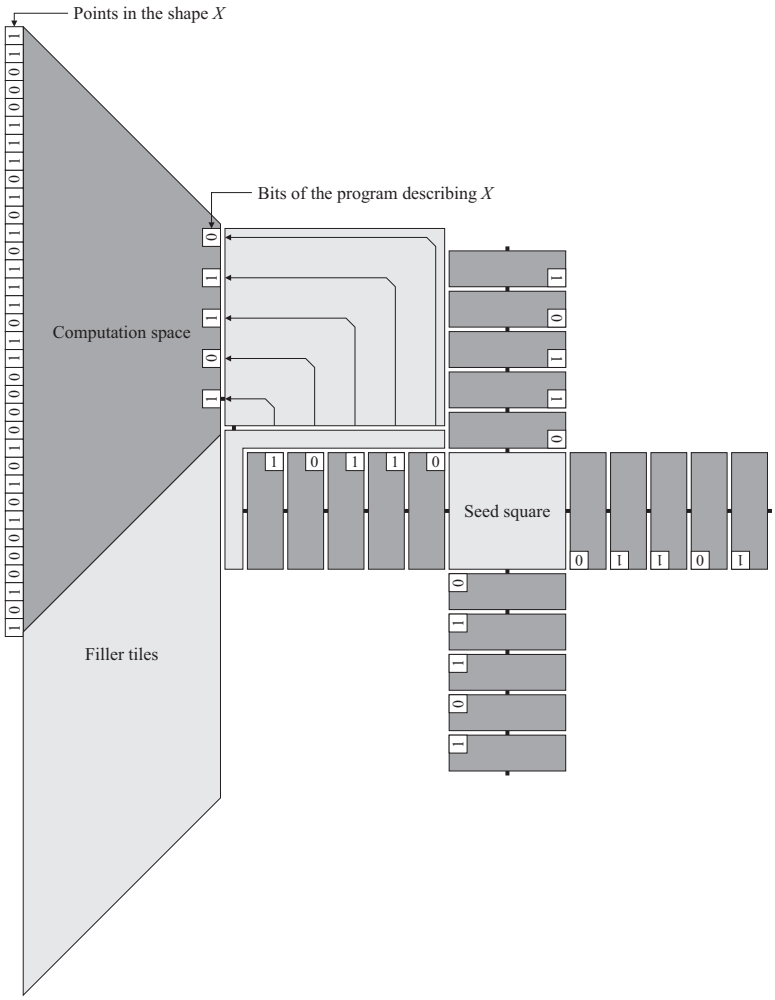
4.2.1 Optimum Temperature Sequences but Unbounded Scaling Factors

In our first construction, we simply combine a portion of the main construction of [54] with the bit-flip gadget of [26].

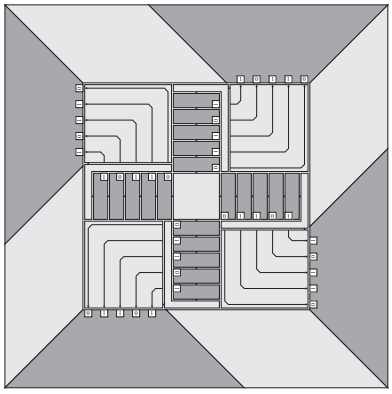
Theorem 18. There exists a tile set T with $|T| = O(1)$ such that, for every finite shape X, there exists $c \in \mathbb{N}$ and a temperature sequence $\langle \tau_j \rangle_{j=0}^{m-1}$ with $m = O(K(X))$ such that $\mathcal{T}^{X^c} = \left(T, \sigma, \langle \tau_j \rangle_{j=0}^{m-1}\right)$ uniquely produces X^c.

Proof. The basic idea is to combine the tile set (of Theorem 3.1) of [26] that assembles an $11 \times 2n$ rectangle (whose top row encodes the bits of an arbitrary binary string of length n) with the portion of the tile set from [54] that does not contain any "seed-frame" or "unpacking" tile types. Thus, given any program π that, when run by U, outputs X as a list of points, we can use bit-flip gadgets to encode a description of π. Then the main construction from [54] can proceed normally at temperature $\tau_{m-1} = 2$. The only (minor) technicality is that we must add additional tile types to ensure that the seed block is properly assembled, which we illustrate in Figure 4.1. Note that the size of this tile set is $O(1)$, i.e., it is independent of the shape being assembled, and the temperature complexity is $O(|\pi|)$. □

www.manaraa.com



(a) Partial seed block: west growing simulation of U on π



(b) A completed seed block

Figure 4.1: In our first construction, the seed square assembles first. Then four identical rectangles simultaneously assemble off each side of the seed square via the corresponding temperature sequence defined in the proof of Theorem 3.1 in [26]. After the self-assembly of the four rectangles, the rotated 'L' structure initiates the assembly of a square in which the bits of π are rotated up and to the left into the computation space. Finally, π is simulated on U in the computation space. After the simulation is done, the shape X is encoded along the border of the seed block. The filler tiles are used to ensure that the seed block is a square.

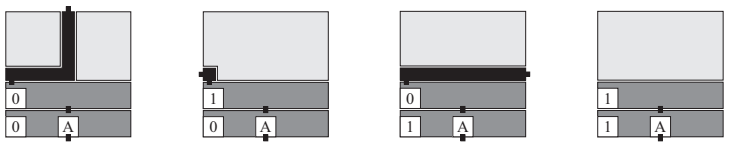
www.manaraa.com

Remark 19. Theorem 18 is tight. In particular, for an algorithmically random string $w = w_0 w_1, \cdots w_{n-1}$, the shape $X(w) = \{(x_0, y_0), (x_1, y_1), \ldots (x_{n-1}, y_{n-1})\}$ defined by $(x_0, y_0) = (0, 0)$, and $(x_{i+1}, y_{i+1}) = (x_{i+1}, y_i + 1)$ if $w_i = 0$, and $(x_i + 1, y_{i+1})$ if $w = 1$ (i.e., a path that goes right if $w_i = 0$ and up if $w_i = 1$), has $K(X(w)) \approx K(w) \geq n$.

Note that in the construction for Theorem 18, the scaling factor can be quite large. Specifically, the scaling factor c depends on the running time of π, whence $c = poly(time(\pi))$ [54]. Also, the scaling factor in the above construction is further inflated (albeit by a constant factor) by the filler tiles in the assembly of the seed block. In a truly nano-scale setting, it is necessary to have a construction in which the scaling factor is always small, or better yet, bounded by a constant independent of the shape being assembled. We now show how to achieve this.

4.2.2 Constant Scaling Factor but Long Temperature Sequences

Recall that for any scaled finite shape X^c, each point in X gets mapped to a $c \times c$ block of points in X^c. In our second construction, we will build a simple *square gadget* that will be responsible for the assembly of each $c \times c$ block in X^c. As a result, the scaling factor c in our second construction will depend entirely on the size of the square gadgets.



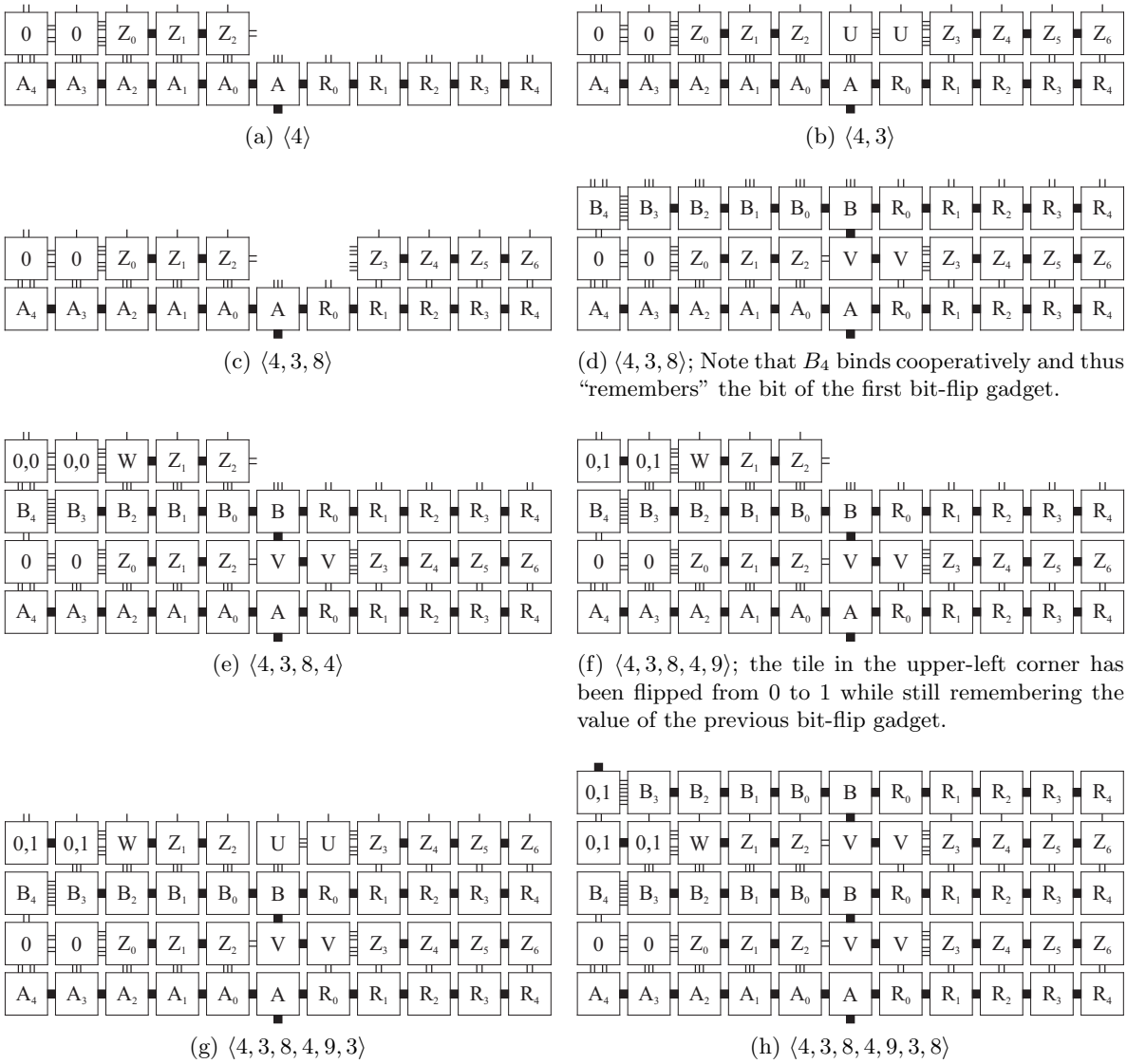
(a) Continue "straight" (b) Turn left (c) Turn right (d) Stop

Figure 4.2: Overview of square gadgets: each square gadget consists of two bit-flip gadgets. The second (a.k.a. upper) bit-flip gadget remembers the value of the first gadget and thus can initiate the correct change in direction. The little black notches on the borders of the first three square gadgets initiate the growth of another (appropriately-rotated) square gadget.

Intuitively, each square gadget consists of two logical components: a lower and an upper half. The lower half of a square gadget is the concatenation of two bit-flip gadgets such that the second bit-flip gadget "remembers" the value of the first. The upper half of each square gadget then places a special output tile along the left, top or right side of the square depending on

www.manaraa.com

the values of the bit-flip gadgets in the lower half. Finally, the special output tile initiates the growth of another (appropriately-rotated) square gadget and the process is repeated for every point in the shape. Figure 4.2 gives an intuitive overview of the four canonical square gadgets. A more detailed example of the self-assembly of a square gadget is shown in Figures 4.3 and 4.4



(a) $\langle 4 \rangle$ (b) $\langle 4, 3 \rangle$

(c) $\langle 4, 3, 8 \rangle$ (d) $\langle 4, 3, 8 \rangle$; Note that B_4 binds cooperatively and thus "remembers" the bit of the first bit-flip gadget.

(e) $\langle 4, 3, 8, 4 \rangle$ (f) $\langle 4, 3, 8, 4, 9 \rangle$; the tile in the upper-left corner has been flipped from 0 to 1 while still remembering the value of the previous bit-flip gadget.

(g) $\langle 4, 3, 8, 4, 9, 3 \rangle$ (h) $\langle 4, 3, 8, 4, 9, 3, 8 \rangle$

Figure 4.3: An example of programming a square gadget to "turn left." We do this by setting the first bit-flip gadget to 0. Then we set the second bit-flip gadget to 1. Note that the latter bit-flip gadget remembers the value of the former bit-flip gadget.

.

Theorem 20. There exists a tile set T with $|T| = O(1)$, such that, for every finite shape X, if

www.manaraa.com

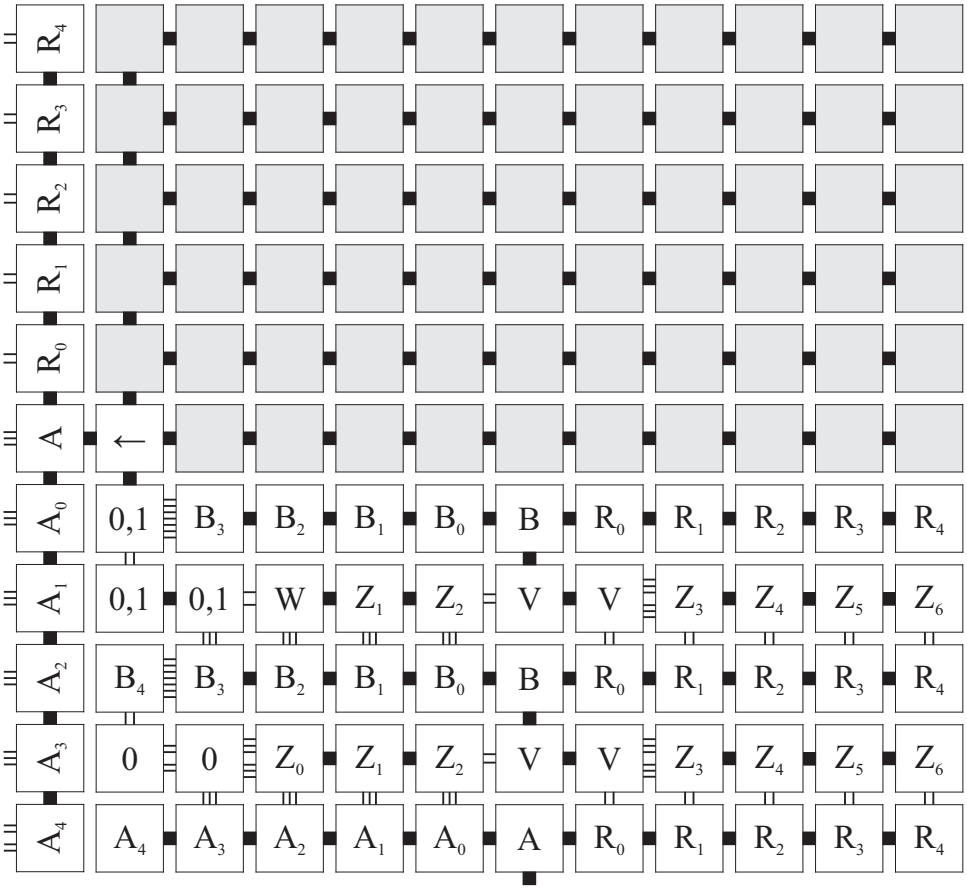


Figure 4.4: $\langle 4, 3, 8, 4, 9, 3, 8 \rangle$; The final result of Figure 4.3. Note the special tile (labeled with '$\leftarrow$') along the left border of the completed square gadget initiated the growth of (the first column of) an appropriately rotated square gadget.

www.manaraa.com

there is a Hamiltonian path C in $G_X^{\#}$, then there exists a temperature sequence $\langle \tau_j \rangle_{j=0}^{m-1}$ with $m = O(|X|)$, such that $\mathcal{T}^{X^{11}} = \left(T, \sigma, \langle \tau_j \rangle_{j=0}^{m-1}\right)$ uniquely produces X^{11}.

Proof. Let $R = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ be the standard 90° counterclockwise rotation matrix. If t is a tile type, then we define $R(t) = t'$, such that, for all $\vec{u} \in U_2$, $\text{str}_{t'}(\vec{u}) = \text{str}_t(R \cdot \vec{u})$ and $\text{col}_{t'}(\vec{u}) = \text{col}_t(R \cdot \vec{u})$. Notice that t' is simply the clockwise rotation of t. Let t_{seed} be the single seed tile type defined in Figure 4.5. Let T be the set of tile types satisfying (1)

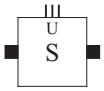


Figure 4.5: The unique seed tile type t_{seed}.

$t_{\text{seed}} \in T$ and (2), for every tile type t that is defined in Figure 4.6, $t, R(t), R^2(t)$ and $R^3(t)$ are all elements of T. This means that T contains four logical copies of the square gadget each having a different "type" of direction. As a final technical matter, we adjust all of the glue colors (excluding the 'A' glue color) on all of the tiles in T such that two tiles can bind if and only if the respective square gadgets to which they belong have the same type of direction.

Let X be an arbitrary finite shape such that there exists a Hamiltonian path

$$C = \left\langle \vec{v}_0, \vec{v}_1, \ldots, \vec{v}_{|X|-1} \right\rangle$$

of $G_X^{\#}$ (here, C is a sequence of vertices). Let $\vec{u}_0 = (0, 1)$. For all $1 \leq i < |X|$, define the unit vector $\vec{u}_i = v_i - v_{i-1}$. Define the temperature sequence $\left\langle \tau_0, \tau_1, \ldots, \tau_{8 \cdot |X|-1} \right\rangle$, where for each

www.manaraa.com

$0 \leq i < |X| - 1$,

$$\tau_{8i} = 4,$$

$$\tau_{8i+1} = \begin{cases} 4 & \text{if } \vec{u}_i = (-R) \cdot \vec{u}_{i-1} \text{ or } \vec{u}_i = \vec{u}_{i-1} \\ 9 & \text{otherwise,} \end{cases}$$

$$\tau_{8i+2} = 3,$$

$$\tau_{8i+3} = 8,$$

$$\tau_{8i+4} = 4,$$

$$\tau_{8i+5} = \begin{cases} 4 & \text{if } \vec{u}_i = R \cdot \vec{u}_{i-1} \text{ or } \vec{u}_i = \vec{u}_{i-1} \\ 9 & \text{otherwise,} \end{cases}$$

$$\tau_{8i+6} = 3, \text{ and}$$

$$\tau_{8i+7} = 8.$$

Finally, let $\tau_{8|X|-8} = 4$, $\tau_{8|X|-7} = 9$, $\tau_{8|X|-6} = 3$, $\tau_{8|X|-5} = 8$, $\tau_{8|X|-4} = 4$, $\tau_{8|X|-3} = 9$, $\tau_{8|X|-2} = 3$, and $\tau_{8|X|-1} = 8$.

Let $\mathcal{T}^{X^{11}} = \left(T, \sigma, \langle \tau_i \rangle_{i=0}^{8 \cdot |X|-1}\right)$. We will now describe how the scaled shape X^{11} self-assembles in $\mathcal{T}^{X^{11}}$.

Assume without loss of generality that if $i = 0$, then $\tau_1 = \tau_5 = 4$ (i.e., $v_1 - v_0 = (0, 1)$). The initial temperature subsequence $\langle \tau_0, \tau_1, \ldots \tau_7 \rangle$ assembles (1) an initial 11×11 "seed" square gadget, denoted S_0, from the single seed tile, and (2) the first 1×11 row (column) of an appropriately-rotated square gadget attached to the top side of the seed gadget.

In general, for each $0 < i < |X| - 1$, the temperature subsequence $\langle \tau_{8i}, \ldots \tau_{8i+7} \rangle$ assembles (1) an appropriately-rotated 11×11 square gadget, denoted S_i, attached to the square gadget assembled in the previous temperature subsequence, and (2) the first 1×11 row (column) of an appropriately-rotated square gadget attached to: the top side of S_i if $\tau_{8i+1} = \tau_{8i+5} = 4$; the left side if $\tau_{8i+1} = 4$ and $\tau_{8i+5} = 9$; or the right side if $\tau_{8i+1} = 9$ and $\tau_{8i+5} = 4$. Intuitively, by our choices of τ_{8i+1} and τ_{8i+5}, we can force the "direction" of self-assembly to follow the Hamiltonian path C. Namely, we set: $\tau_{8i+1} = \tau_{8i+5} = 4$ in order to *continue* straight (relative to the current direction of self-assembly); $\tau_{8i+1} = 4$ and $\tau_{8i+5} = 9$ to initiate a relative *left*

www.manaraa.com

turn; $\tau_{8i+1} = 9$ and $\tau_{8i+5} = 4$ to initiate a relative *right turn*; and $\tau_{8i+1} = 9$ and $\tau_{8i+5} = 9$ to *halt* self-assembly.

Finally, we have $\tau_{8|X|-7} = \tau_{8|X|-3} = 9$, and the last temperature subsequence assembles the final 11×11 (halting) square gadget attached to the square gadget assembled in the previous temperature subsequence.

Since each square gadget is an 11×11 square, and C is a Hamiltonian path of $G_X^{\#}$, we have that X^{11} self-assembles in $\mathcal{T}^{X^{11}}$. An intuitive illustration of this process is depicted in Figures 4.7(b) and 4.7(c). Also, Figures 4.3 and 4.4 in the technical appendix give a detailed example of how to program a square gadget to turn left. A formal proof of the fact that $\mathcal{T}^{X^{11}}$ uniquely produces X^{11} is, although tedious, straightforward, and therefore omitted. □

In our second construction, we will encode a Hamiltonian path of a particular finite shape X into a temperature sequence in order to assemble the scaled-up version of X. Unfortunately, not all shapes have Hamiltonian paths, which might suggest that this approach is doomed to fail. Lucky for us, however, we have the following technical lemma.

Lemma 21. If X is a finite shape, then there exists a Hamiltonian cycle C in $G_{X^2}^{\#}$.

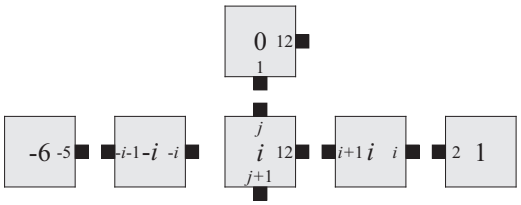
Proof. Note that, in this proof, we will think of Hamiltonian cycles as sequences of edges. For every finite shape X, define the set $\mathcal{B}(X) = \{(x, y) \in X \mid (\exists \vec{u} \in U_2)$ satisfying $(x, y) + \vec{u} \notin X\}$. One can think of the set $\mathcal{B}(X)$ as the set of all points from which it is possible to "get away from" the shape X in one step, that is (in some sense), the points along the "border" of X. In what follows, we will prove that there exists a Hamiltonian cycle C in $G_{X^2}^{\#}$ with the following property P:

> For every $(w, x), (y, z) \in \mathcal{B}\left(X^2\right)$ with $(y, z) - (w, x) \in U_2$, such that $\left(\left\lfloor \frac{y}{2} \right\rfloor, \left\lfloor \frac{z}{2} \right\rfloor\right) = \left(\left\lfloor \frac{w}{2} \right\rfloor, \left\lfloor \frac{x}{2} \right\rfloor\right)$, the edge $((w, x), (y, z)) \in C$.

Our proof is by induction on $|X|$. For the base case, we have $|X| = 1$, and it is routine to verify that $G_{X^2}^{\#}$ has a Hamiltonian cycle C satisfying property P.

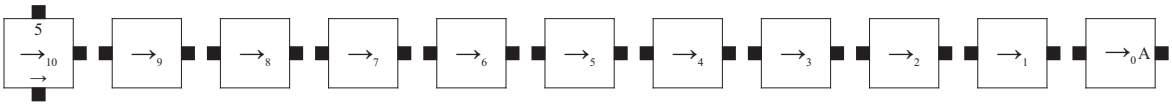
For the inductive case, let X be a shape with $|X| = k + 1$. We will show that $G_{X^2}^{\#}$ has a Hamiltonian cycle C satisfying property P. Let $\vec{x} \in X$ be an arbitrary point such that $X - \{\vec{x}\}$ is a shape. Then define the shape $Y = X - \{\vec{x}\}$. Since Y is a shape with $|Y| = k$, the

www.manaraa.com

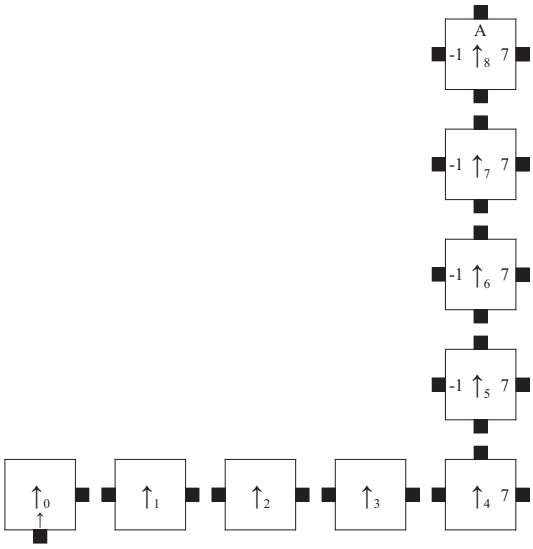


(a) Filler tiles. Here, $i \in \{-6, \ldots, -1, 2, \ldots, 12\}$, and $1 \leq j \leq 4$

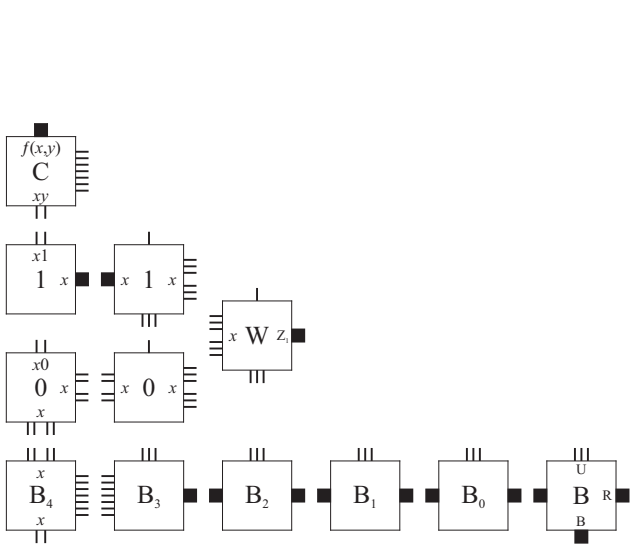
(b) The '←' tile initiates the growth of another square gadget "west" of the current square gadget; the other tile does not initiate any subsequent square gadgets



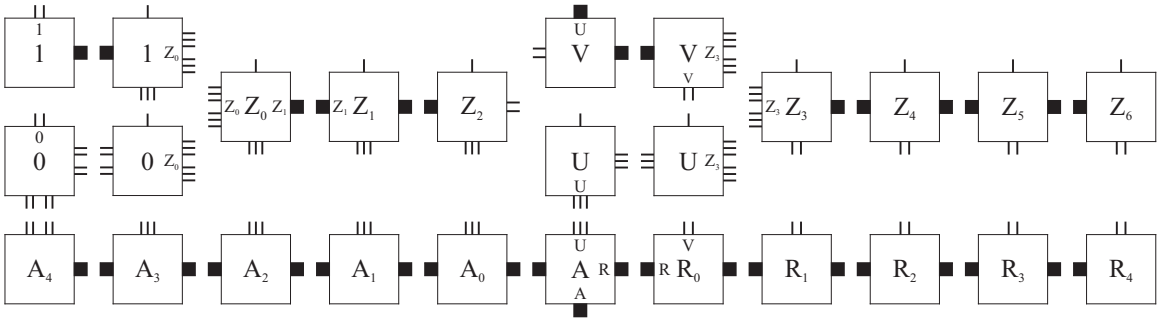
(c) Tiles that initiate growth of another square gadget "east" of the current square gadget



(d) Tiles that initiate growth of another square gadget directly "north" of the current square gadget



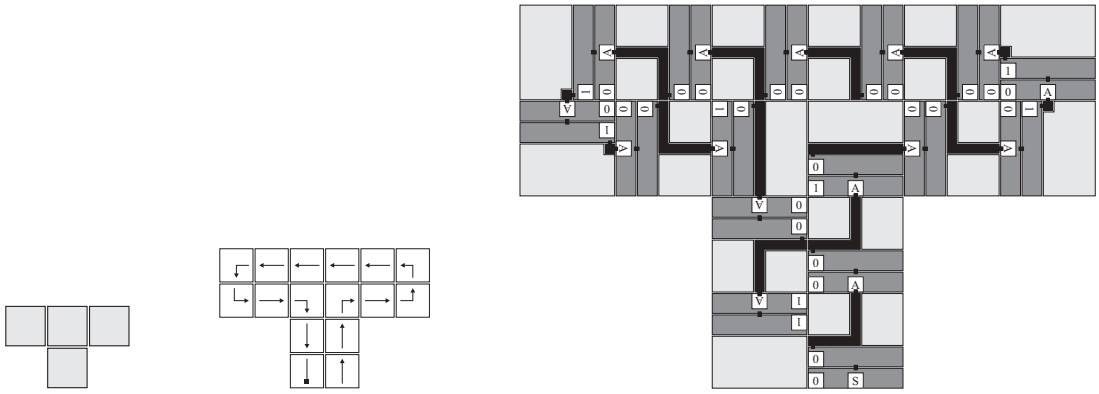
(e) Tiles for the second bit-flip gadget. Here $x, y \in \{0, 1\}$, and $f : \{0, 1\}^2 \longrightarrow \{\uparrow, \leftarrow, \rightarrow, \text{stop}\}$ is defined as follows: $f(0, 0) = \uparrow$, $f(0, 1) = \leftarrow$, $f(1, 0) = \rightarrow$, and $f(1, 1) = \text{stop}$.



(f) Tiles for the first bit-flip gadget. They are essentially the same as the those of the bit-flip gadget in [26]

Figure 4.6: Tile types for the square gadget. The thick notches represent strength 9 bonds.

www.manaraa.com



(a) Example shape X

(b) Hamiltonian path of $G^{\#}_{X^2}$ given by the proof of Lemma 21.

(c) Self-assembly of X^{22}. The 'S' tile is the single seed tile; self-assembly proceeds from this initial square gadget and follows the Hamiltonian path given in (b). Notice that the first three square gadgets have the same type of direction, while the fourth square gadget is rotated 90° clockwise.

Figure 4.7: Overview of second construction for Theorem 22.

induction hypothesis tells us that $G^{\#}_{Y^2}$ has a Hamiltonian cycle $D = \langle e_0, e_1, \ldots e_{k-1} \rangle$ satisfying property P. We will use D to construct a Hamiltonian cycle C in $G^{\#}_{X^2}$ having property P as follows. Let $\vec{a}, \vec{b}, \vec{c}, \vec{d} \in X^2$ be such that $\left\{\vec{a}, \vec{b}, \vec{c}, \vec{d}\right\} = \left\{(x, y) \mid \left(\left\lfloor \frac{x}{2} \right\rfloor, \left\lfloor \frac{y}{2} \right\rfloor\right) = \vec{x}\right\}$. Since D satisfies property P, there exist points $\vec{p}, \vec{q} \in Y^2$ such that, there exists $\vec{u} \in U_2$ satisfying $\vec{a} + \vec{u} = \vec{p}$, $\vec{d} + \vec{u} = \vec{q}$, and $(\vec{p}, \vec{q}) \in D$. Suppose that $e_j = (\vec{p}, \vec{q})$ for some $0 \leq j < k$. Finally, let $C = \left\langle e_0, e_1, \ldots, e_{j-1}, (\vec{p}, \vec{a}), \left(\vec{a}, \vec{b}\right), \left(\vec{b}, \vec{c}\right), \left(\vec{c}, \vec{d}\right), \left(\vec{d}, \vec{q}\right), e_{j+1}, \ldots, e_{k-1} \right\rangle$. It is clear that C is a Hamiltonian cycle in $G^{\#}_{X^2}$ having property P. □

Lemma 21 says that, if we are comfortable with doubling the scaling factor, we can always use a Hamiltonian cycle to direct the self-assembly of an arbitrary scaled shape.

Theorem 22. There exists a tile set T with $|T| = O(1)$ such that, for every finite shape X, there exists a temperature sequence $\langle \tau_j \rangle_{j=0}^{m-1}$ with $m = O(|X|)$, such that $\mathcal{T}^{X^{22}} = \left(T, \sigma, \langle \tau_j \rangle_{j=0}^{m-1}\right)$ uniquely produces X^{22}.

Proof. The theorem follows by Theorem 20, Lemma 21, and the simple observation that, for any finite shape X,

$$\left(X^2\right)^{11} = X^{2 \cdot 11} = X^{22}.$$

www.manaraa.com

□

4.3 Impossibility of Self-Assembly of Arbitrary Shapes with $O(1)$ Tile Types

At this point, a natural question might be the following: Is the scaling factor in both of our constructions necessary? This question can be stated formally as follows.

Question 23 (Kao and Schweller [26])**.** Does there exist a tile set T, with $|T| = O(1)$, such that for every finite shape X, there exists a temperature sequence $\langle \tau_j \rangle_{j=0}^{m-1}$ such that $\mathcal{T} = \left(T, \sigma, \langle \tau_j \rangle_{j=0}^{m-1}\right)$ uniquely produces X?

In the remainder of this section, we prove that the answer to Question 23 is "no." Recall that if $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ is an assembly sequence in $\mathcal{T}$ and $\vec{m} \in \mathbb{Z}^2$, then the $\vec{\alpha}$-*index* of $\vec{m}$ is $i_{\vec{\alpha}}(\vec{m}) = \min\{i \in \mathbb{N} \mid \vec{m} \in \text{dom } \alpha_i\}$. That is, the $\vec{\alpha}$-index of $\vec{m}$ is the time at which any tile is first placed at location $\vec{m}$ by $\vec{\alpha}$. For each location $\vec{m} \in \bigcup_{0 \leq i < l} \text{dom } \alpha_i$, define the set

$$\text{IN}^{\vec{\alpha}}(\vec{m}) = \left\{ \vec{u} \in U_2 \;\middle|\; i_{\vec{\alpha}}(\vec{m} + \vec{u}) < i_{\vec{\alpha}}(\vec{m}) \text{ and } \text{str}_{\alpha_{i_{\vec{\alpha}}(\vec{m})}}(\vec{m}, \vec{u}) > 0 \right\}.$$

Intuitively, the set $\text{IN}^{\vec{\alpha}}(\vec{m})$ is the set of sides on which the *first* tile that $\vec{\alpha}$ places at location $\vec{m}$ *initially* binds. We now have the machinery to prove the following result.

Theorem 24. For every tile set T, there exists a finite shape $X \subseteq \mathbb{Z}^2$ such that for each temperature sequence $\langle \tau_i \rangle_{i=0}^{m-1}$, $\mathcal{T} = \left(T, \sigma, \langle \tau_i \rangle_{i=0}^{m-1}\right)$ does not uniquely produce X.

Proof. Fix a set of tile types T, and let $X = \{0, \ldots, |T|\} \times \{0\}$. We will show that, given any temperature sequence $\langle \tau_i \rangle_{i=0}^{m-1}$, the tile system $\mathcal{T} = \left(T, \sigma, \langle \tau_i \rangle_{i=0}^{m-1}\right)$ does not uniquely produce X. To get a contradiction, assume that $\mathcal{T}$ uniquely produces X and let α be the unique assembly satisfying $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$. Since $|X| > |T|$, there must exist $\vec{x}_p, \vec{x}_q \in X$ such that $\alpha(\vec{x}_p) = \alpha(\vec{x}_q)$. By the fact that $\mathcal{T}$ uniquely produces X, we know that, by definition, every assembly sequence in $\mathcal{T}$ is finite. Therefore, it suffices to exhibit an infinite assembly sequence in $\mathcal{T}$. The next fact is an easy consequence of the pigeonhole principle.

www.manaraa.com

Fact 1. If $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ is an assembly sequence in $\mathcal{T}$ such that, for all $0 \leq i < k$, dom $\alpha_i \subseteq X$ and dom res$(\vec{\alpha}) = X$, then there exists an infinite assembly sequence $\vec{\alpha}'$ in $\mathcal{T}$.

For each assembly sequence $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ in $\mathcal{T}$ such that for some $0 \leq i < k$, dom $\alpha_i - X \neq \varnothing$, let $\vec{y}_{\vec{\alpha}}$ be the unique point $\vec{y}_{\vec{\alpha}} \notin X$ such that, for all $\vec{z} \notin X \cup \{\vec{y}_{\vec{\alpha}}\}$, $i_{\vec{\alpha}}(\vec{y}_{\vec{\alpha}}) < i_{\vec{\alpha}}(\vec{z})$. Intuitively, the point $\vec{y}_{\vec{\alpha}}$ is the location of the first tile that $\vec{\alpha}$ places at any point not in X. The next fact gives sufficient conditions for the existence of an infinite assembly sequence in $\mathcal{T}$.

Fact 2. Let $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ be an assembly sequence in $\mathcal{T}$ such that for some $0 \leq i < k$, dom $\alpha_i - X \neq \varnothing$, and $\vec{u}$ be the unique vector satisfying $\vec{u} \in \text{IN}^{\vec{\alpha}}(\vec{y}_{\vec{\alpha}})$. If $\text{str}_{\alpha_{i_{\vec{\alpha}}}(\vec{y}_{\vec{\alpha}})}(\vec{y}_{\vec{\alpha}}, \vec{u}) < \tau_{m-1}$ and, for every $\vec{x} \in X$,

$$i_{\vec{\alpha}}(\vec{x}) < i_{\vec{\alpha}}(\vec{y}_{\vec{\alpha}}) \Rightarrow \alpha_{i_{\vec{\alpha}}(\vec{x})}(\vec{x}) = \begin{cases} \alpha(\vec{x} - (\vec{x}_q - \vec{x}_p)) & \text{if } \vec{x} - \vec{x}_q \in X \\ \alpha(\vec{x}) & \text{otherwise,} \end{cases}$$

then there exists an infinite assembly sequence $\vec{\alpha}'$ in $\mathcal{T}$.

Proof. (*of Fact 2*) Let $(x_0, 0) = \vec{x} \in X$ be the location of the rightmost tile in the assembly $\alpha_{i_{\vec{\alpha}}(\vec{y}_{\vec{\alpha}})}$. We can define an *infinite* assembly sequence $\vec{\alpha}' = (\alpha'_i \mid 0 \leq i < k)$ as follows. For $0 \leq i \leq i_{\vec{\alpha}}(\vec{y}_{\vec{\alpha}})$, let $\alpha'_i = \alpha_i$. Notice that at this point, the temperature of the current temperature phase must be less than τ_{m-1} since $\text{str}_{\alpha_{i_{\vec{\alpha}}}(\vec{y}_{\vec{\alpha}})}(\vec{y}_{\vec{\alpha}}, \vec{u}) < \tau_{m-1}$. This means that we can keep placing tiles sequentially at points in X (because α is τ_{m-1}-stable, whence every bond between adjacent tiles in α must be at least this strong). For $i_{\vec{\alpha}}(\vec{y}_{\vec{\alpha}}) \leq i < (|X|-1) + i_{\vec{\alpha}}(\vec{y}_{\vec{\alpha}}) - x_0$, let α'_{i+1} be the assembly obtained from α'_i by placing a tile of type $\alpha(\vec{x} + (i - i_{\vec{\alpha}}(\vec{y}_{\vec{\alpha}}) + 1, 0) - (\vec{x}_q - \vec{x}_p))$ at the point $\vec{x} + (i - i_{\vec{\alpha}}(\vec{y}_{\vec{\alpha}}) + 1, 0)$ if $\vec{x} + (i - i_{\vec{\alpha}}(\vec{y}_{\vec{\alpha}}) + 1, 0) - \vec{x}_q \in X$, and $\alpha(\vec{x} + (i - i_{\vec{\alpha}}(\vec{y}_{\vec{\alpha}}) + 1, 0))$ otherwise. At this point, a tile has been placed at every point in X. But why stop there? For $i \geq (|X| - 1) + i_{\vec{\alpha}}(\vec{y}_{\vec{\alpha}}) - x_0 - 1$, let α'_{i+1} be obtained from α'_i by placing a tile of type $\alpha'_i((i + 1 - (i_{\vec{\alpha}}(\vec{y}_{\vec{\alpha}}) - x_0 - 1), 0) - (\vec{x}_q - \vec{x}_p))$ at the point $(i + 1 - (i_{\vec{\alpha}}(\vec{y}_{\vec{\alpha}}) - x_0 - 1), 0)$. A routine induction argument can be used to show that $\vec{\alpha}'$ is an infinite (eventually) periodic assembly sequence in $\mathcal{T}$. □

www.manaraa.com

Fact 3. There exists an assembly sequence $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ in $\mathcal{T}$ such that for some $0 \leq i < k$, $\text{dom}\, \alpha_i - X \neq \varnothing$ and, for every $\vec{x} \in X$,

$$i_{\vec{\alpha}}(\vec{x}) < i_{\vec{\alpha}}(\vec{y}_{\vec{\alpha}}) \Rightarrow \alpha_{i_{\vec{\alpha}}(\vec{x})}(\vec{x}) = \begin{cases} \alpha(\vec{x} - (\vec{x}_q - \vec{x}_p)) & \text{if } \vec{x} - \vec{x}_q \in X \\ \alpha(\vec{x}) & \text{otherwise.} \end{cases}$$

Fact 3 follows by the simple observations that (1) every assembly sequence in $\mathcal{T}$ is finite (because $\mathcal{T}$ uniquely produces the shape X), and (2) until some tile is placed at some point outside of X, every tile must bind via a single input side.

Let $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ be the assembly sequence in $\mathcal{T}$ given in Fact 3. Suppose that the first tile that $\vec{\alpha}$ places at $\vec{y}_{\vec{\alpha}} \notin X$ binds to the single seed tile. In this case, the tile that is placed at $\vec{y}_{\vec{\alpha}}$ must bind via a single bond having strength less than τ_{m-1}, because otherwise (if it were to bind with strength at least τ_{m-1}), it would be possible to place a tile at $\vec{y}_{\vec{\alpha}}$ before finishing the final temperature phase (contradicting the the fact that $\mathcal{T}$ uniquely produces X). If the first tile that $\vec{\alpha}$ places at $\vec{y}_{\vec{\alpha}}$ binds to the single seed tile with strength less than t_{m-1}, then Fact 2 gives us an infinite assembly sequence $\vec{\alpha}'$ in $\mathcal{T}$, which is a contradiction.

Therefore, the first tile that $\vec{\alpha}$ places at the point $\vec{y}_{\vec{\alpha}}$ *cannot* bind to the seed tile. Let $\vec{u}$ be the unique vector satisfying $\vec{u} \in \text{IN}^{\vec{\alpha}}(\vec{y}_{\vec{\alpha}})$. Suppose that $\text{str}_{\alpha_{i_{\vec{\alpha}}}(\vec{y}_{\vec{\alpha}})}(\vec{y}_{\vec{\alpha}}, \vec{u}) \geq \tau_{m-1}$. If $(\vec{y}_{\vec{\alpha}} + \vec{u}) - \vec{x}_q \notin X$, then it would be possible to place some tile at the point $\vec{y}_{\vec{\alpha}} + \vec{u}$ during the final temperature phase because $\alpha_{i_{\vec{\alpha}}(\vec{y}_{\vec{\alpha}}+\vec{u})}(\vec{y}_{\vec{\alpha}} + \vec{u}) = \alpha(\vec{y}_{\vec{\alpha}} + \vec{u})$, which is a contradiction. On the other hand, if $(\vec{y}_{\vec{\alpha}} + \vec{u}) - \vec{x}_q \in X$, then it would be possible to place some tile at the point $(\vec{y}_{\vec{\alpha}^*} + \vec{u}) - (\vec{x}_q - \vec{x}_p)$ during the final temperature phase because $\alpha_{i_{\vec{\alpha}}(\vec{y}_{\vec{\alpha}}+\vec{u})}(\vec{y}_{\vec{\alpha}} + \vec{u}) = \alpha((\vec{y}_{\vec{\alpha}} + \vec{u}) - (\vec{x}_q - \vec{x}_p))$, which is a contradiction. Therefore, it must be that $\text{str}_{\alpha_{i_{\vec{\alpha}}}(\vec{y}_{\vec{\alpha}})}(\vec{y}_{\vec{\alpha}}, \vec{u}) < \tau_{m-1}$, and Fact 2 gives us an infinite assembly sequence $\vec{\alpha}'$ in $\mathcal{T}$, which is a contradiction. □

In other words, Theorem 24 says that the scaling factor c in each of our constructions is necessary – regardless of how many temperature changes is allowed.

4.4 Conclusion

In this chapter, we showed how to reduce the tile complexity for the self-assembly of arbitrary scaled shapes in the multiple temperature model. We developed two general purpose

www.manaraa.com

tile sets capable of assembling scaled-up versions of arbitrary finite shapes through appropriately chosen sequences of non-negative integer temperatures. While our first construction assembled shapes via short (asymptotically Kolmogorov-optimum) temperature sequences, the scaling factor grows (unboundedly) with the size of the shape being assembled. In contrast, our second construction assembled shapes via long temperature sequences but with a constant "universal" scaling factor in the sense that it is the same for every shape. We then proved that, for every constant-size tile set T, there is some finite shape that T cannot uniquely produce via any temperature sequence, which implies the necessity of the scaling factor in both of our constructions. A natural direction for future theoretical research in the multiple temperature model is the next question, which asks whether we can simultaneously optimize the two criteria of a short temperature sequence and a constant scaling factor, which were optimized separately in the constructions of this chapter.

Question 25. Does there exist a tile set T with $|T| = O(1)$ and $c \in \mathbb{N}$, such that, for every shape X, there exists a temperature sequence $\langle \tau_j \rangle_{j=0}^{m-1}$ with $m = O(K(X))$, such that $\mathcal{T}^{X^c} = \left(T, \sigma, \langle \tau_j \rangle_{j=0}^{m-1}\right)$ uniquely produces X^c?

www.manaraa.com

CHAPTER 5. Intrinsic Universality in the Abstract Tile Assembly Model

This chapter is joint work with David Doty, Jack Lutz, Matthew Patitz and Damien Woods and was published in [18].

5.1 Introduction

The development of DNA tile self-assembly has moved nanotechnology closer to the goal of engineering useful systems that assemble themselves from molecular components. Since Seeman's pioneering work in the 1980s [51], many laboratory experiments have shown that DNA tiles can be designed to spontaneously assemble with one another into desired structures [48]. As physical and mathematical error-suppression techniques improve [10, 22, 35, 53, 55], this molecular programming of matter will become practical at ever larger scales.

The Tile Assembly Model, developed by Winfree [47,58], is a discrete mathematical model of DNA tile self-assembly that enables us to explore the potentialities and limitations of this kind of molecular programming. It is essentially an "effectivization" of classical Wang tiling [56] in which the fundamental components are un-rotatable, but translatable square "tile types" whose sides are labeled with glue "colors" and "strengths." Two tiles that are placed next to each other *interact* if the glue colors on their abutting sides match, and they *bind* if the strength on their abutting sides matches with total strength at least a certain ambient "temperature." Extensive refinements of the abstract Tile Assembly Model were given by Rothemund and Winfree in [46, 47]. (Consult the technical appendix for full details of the abstract Tile Assembly Model.) The model deliberately oversimplifies the physical realities of self-assembly, but Winfree proved that it is Turing universal [58], implying that self-assembly can be algorithmically directed.

www.manaraa.com

In this chapter we investigate whether the Tile Assembly Model is capable of a much stronger notion of universality where the goal is to give a single tile assembly system that simulates the behavior of any other tile assembly system. We give a tile assembly system that is capable of simulating a very wide class of tile systems, including itself. Our notion of simulation is inspired by, but somewhat stronger than, intrinsic universality in cellular automata [4, 21, 38–40]. In our construction a simulated tile assembly system is encoded in a seed assembly of the simulating system. This encoding is done in a very simple (logspace computable) way. The seed assembly then grows to form an assembly that is a re-scaled (larger) version of the simulated assembly, where each tile in the latter is represented by a supertile (square of tiles) in the simulator. Not only this, but each of the possible (nondeterministically chosen) assembly sequences of the simulated tile system is modeled by a possible assembly sequence in the simulating system (also nondeterministically chosen). The latter property of our system is important and highlights one way in which this work distinguishes itself from other notions of intrinsic universality found in the cellular automata literature: not only do we want to simulate the final assembly but we also want the simulator to have the ability to dynamically simulate each of the valid growth processes that could lead to that final assembly.

A second distinguishing property of our universal tile set is that it simulates nondeterministic choice in a "fair" way. An inherent feature of the Tile Assembly Model is the fact there are often multiple (say k) tiles that can go into any one position in an assembly sequence, and one of these k is nondeterministically chosen. One way to simulate this feature is to nondeterministically choose which of k supertiles should grow in the analogous (simulated) position. However, due to the size blowup in supertiles caused by encoding an arbitrary-sized simulated tile set into a fixed-sized universal tile set, it seems that we need to simulate one nondeterministic choice by using a sequence of nondeterministic choices within the supertile. Interpreting the nondeterministic choice to be made according to uniform random selection, if the selection by the simulating tile set is implemented in a naïve way, this can lead to unfair selection: when selecting 1 supertile out of k, some supertiles are selected with extremely low probability. To get around this problem, our system uses a random number selector that chooses a random tile

www.manaraa.com

with probability $\Theta(1/k)$ and so we claim that we are simulating nondeterminism in a "fair" way.

Thirdly, the Tile Assembly Model has certain geometric constraints that are not seen in cellular automata, and this adds some difficulty to our construction. Existing techniques for constructing intrinsically universal cellular automata are not directly applicable to tile assembly. For example, when a tile is placed at a position, that position can not be reused for further "computation" and this presents substantial difficulties when trying to fit the various components of our construction into a supertile. Each supertile encodes the entire simulated tile set and has the functionality to propagate this information to other (yet to be formed) supertiles. Not only this, each supertile must decide which tile placement to simulate, whilst making (fair) nondeterministic choices if necessary. Finally, each supertile should correctly propagate (output) sides that are consistent with the chosen supertile. We give a number of figures to illustrate how these goals were met within the geometric constraints of the model.

Our main result presented in this chapter is, in some sense, a continuation of some previous results in self-assembly. For instance, Soloveichik and Winfree [54] exhibit a beautiful connection between the Kolmogorov complexity of a finite shape X and the minimum number of tile types needed to assemble X. It turns out that their construction can be made to be "universal" in the following sense: there exists a tile set T, such that for every "temperature 1" tile assembly system that produces a finite shape whose underlying binding graph is a spanning tree, T simulates the given temperature 1 tile system with a corresponding blow-up in the scale. Note that this method restricts the simulated tile system to be temperature 1, i.e., a non-cooperative tile assembly system, which are conjectured [20] to produce "simple" shapes and patterns in the sense of Presburger arithmetic [41].

A similar result, recently discovered by Demaine, Demaine, Fekete, Ishaque, Rafalin, Schweller, and Souvaine [16], established the existence of a general-purpose "staged-assembly" system that is capable of simulating any temperature 1 tile assembly system that produces a "fully connected" finite shape. Note that, in this construction, the scaling factor is proportional to $O(\log |T|)$, where T is the simulated tile set. This construction has the desirable

www.manaraa.com

property that the set of tile types belonging to the simulator is general purpose (i.e., the size of the simulator tile set is independent of the to-be-simulated tile set) and all of the information needed to carry out the simulation is, in some sense, encoded in a sequence of laboratory steps. An open question in [16] is whether or not their construction can be augmented to handle temperature 2 tile assembly systems.

Our construction is general enough to be able to simulate powerful and interesting tile sets, yet sufficiently simple so that it actually belongs to the class of tile assembly systems that it can simulate, a class we term *locally consistent*. Systems in this class have the properties that each tile binds with exactly strength 2, and there are no glue mismatches in any producible assembly. This captures a wide class of tile assembly systems, including counters, square-builders and other shape-building tile assembly systems, and the tile assembly systems described in [2, 31, 47, 54]. Modulo re-scaling, our universal tile set can be said to display the characteristics of the entire collection of tile sets in its class. Our construction is a *direct simulation* in that the technique does not involve the simulation of intermediate models (such as circuits or Turing machines), which have been used in intrinsically universal cellular automata constructions [40].

One of the nice properties of intrinsic universality [40] is that it provides a clear definition that facilitates proofs that a given tile set is not universal. We leave as an open problem the intrinsic universality status of the Tile Assembly Model in its full generality.

Lafitte and Weiss [28–30] have also studied universality in the related model of Wang tiling [56]. Some of their definitions, particularly in [29], are similar to our definitions of simulation and universality, and also to those of Ollinger [40]. However, Wang tiling is not a model of self-assembly, as it is concerned with the ability of finite tile sets to tile the whole plane (with no mismatches), without regard to the *process* by which these tiles are placed. What is important is simply the existence of some valid tiling. In the Tile Assembly Model, which takes the order in which tiles are placed, one by one, into account, it must be shown that not only is there a sequence by which tiles could be individually and stably added to form the output assembly, but that *every* possible such sequence leads to the desired output. Furthermore, in the Tile Assembly Model, a tile addition can be valid even if it causes mismatches as long as

www.manaraa.com

it is stable.

Most attempts to adapt the constructions of Wang tiling studies (such as those in [28–30]) to self-assembly result in a tile assembly system in which many junk assemblies are formed due to incorrect nondeterministic choices being made that arrest any further growth and/or result in assemblies which are inconsistent with the desired output assembly. We therefore require novel techniques to ensure that no nondeterminism is introduced, other than that already present in the tile system being simulated, and that the only produced assemblies are those that represent the intended result or valid partial progress toward it.

5.2 Intrinsic Universality in Self-Assembly

In this section, we define our notion of intrinsic universality of tile assembly systems. It is inspired by, but distinct from, similar notions for cellular automata [40]. Where appropriate, we identify where some part of our definition differs from the "corresponding" parts in [40], typically due to a fundamental difference between the abstract Tile Assembly Model and cellular automata models.

Intuitively, a tile set U is universal for a class $\mathfrak{C}$ of tile assembly systems if U can "simulate" any tile assembly system in $\mathfrak{C}$, where we use an appropriate seed assembly to give a tile assembly system $\mathcal{U}$. U is intrinsically universal if the simulation of $\mathcal{T}$ by $\mathcal{U}$ can be done according to a simple "block substitution scheme" where equal-size square blocks of tiles in assemblies produced by $\mathcal{U}$ represent tiles in assemblies produced by $\mathcal{T}$. Furthermore, since we wish to simulate the entire process of self-assembly, and not only the final result, it is critical that the simulation be such that the "local transition rules" involving intermediate producible (and nonterminal) assemblies of $\mathcal{T}$ be faithfully represented in the simulation.

In the subsequent definitions, given two partial functions f, g, we write $f(x) = g(x)$ if f and g are both defined and equal on x, or if f and g are both undefined on x. Let $c, c' \in \mathbb{N}$, let $[c : c']$ denote the set $\{c, c+1, \ldots, c'-1\}$, and let $[c]$ denote the set $[0 : c] = \{0, 1, \ldots, c-1\}$, so that $[c]^2$ forms a $c \times c$ square with the origin as the lower-left corner.

The natural analog of a configuration of a cellular automaton is an assembly of a tile

www.manaraa.com

assembly system. However, unlike cellular automata in which every cell has a well-defined state, in tile assembly, there is a fundamental difference between a point being empty space and being occupied by a tile. Therefore we keep the convention of representing an assembly as a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$ (for some tile set T), rather than treating empty space as just another type of tile.

Let $\mathcal{T} = (T, \sigma_{\mathcal{T}}, \tau)$ and $\mathcal{S} = (S, \sigma_{\mathcal{S}}, \tau)$ be tile assembly systems. For simplicity, assume that $\sigma_{\mathcal{T}}(0,0)$ is defined, and $\sigma_{\mathcal{T}}$ is undefined on $\mathbb{Z}^2 - \{(0,0)\}$ (i.e., $\mathcal{T}$ is singly-seeded with the seed tile placed at the origin). We will use this assumption of a single seed throughout the paper, but it is not strictly necessary and is only used for simplicity of discussion. Define a *representation function* to be a partial function of the form $r : ([c]^2 \dashrightarrow S) \dashrightarrow T$. That is, r takes a pattern $p : [c]^2 \dashrightarrow S$ of tile types from S painted onto a $c \times c$ square (with locations at which p is undefined representing empty space), and (if r is defined for input p) gives a single tile type from T. Intuitively, r tells us how to interpret $c \times c$ blocks within assemblies of $\mathcal{S}$ as single tiles of T. We write REPR for the set of all representation functions.[1]

We say $\mathcal{S}$ *(intrinsically) simulates* $\mathcal{T}$ *with resolution loss* c if there exists a representation function $r : ([c]^2 \dashrightarrow S) \dashrightarrow T$ such that the following conditions hold.

1. dom $\sigma_{\mathcal{S}} \subseteq [c]^2$ and $r(\sigma_{\mathcal{S}}) = \sigma_{\mathcal{T}}(0,0)$, i.e., the seed assembly of $\mathcal{S}$ represents the seed of $\mathcal{T}$.

2. For every producible assembly $\alpha_{\mathcal{T}} \in \mathcal{A}[\mathcal{T}]$ of $\mathcal{T}$, there is a producible assembly $\alpha_{\mathcal{S}} \in \mathcal{A}[\mathcal{S}]$ of $\mathcal{S}$ such that, for every $x, y \in \mathbb{Z}$,

$$r\left((\alpha_{\mathcal{S}} \upharpoonright ([cx : c(x+1)] \times [cy : c(y+1)])) + (-cx, -cy)\right) = \alpha_{\mathcal{T}}(x, y).$$

[1]Here, r is analogous to Ollinger's ϕ function for mixautomata which maps a state s of the simulated cellular automaton to a set of (blocks of) states in the simulating cellular automaton, however r is defined to map tile block patterns of the simulating tile system to the simulated tile system. So we have phrased our representation in "inverse" terms; i.e., a many-to-one function. In the case of cellular automata, every cellular automaton that is universal with multiple state representations ("intrinsically universal with respect to mixed bulking" in Ollinger's terminology) is known also to be universal with unique state representations ("intrinsically universal with respect to injective bulking"). In contrast, we do not know how to alter our construction so that each tile in the simulated TAS will have a unique representation as a block in the simulating TAS, since our block assembly depends on the direction(s) of the input sides of the tile, and a single tile type may not have have a unique set of input sides throughout the assembly process. This difference between tile assembly and cellular automata, in which tiles may not have a fixed "neighborhood", may imply that this difference is fundamental and not an artifact of our construction.

www.manaraa.com

That is, the $c \times c$ block at (relative) position (x, y) (relative to the other $c \times c$ blocks; the absolute position is (cx, cy)) of assembly $\alpha_{\mathcal{S}}$ represents the tile type at (absolute) position (x, y) of assembly $\alpha_{\mathcal{T}}$. In this case, write $r^*(\alpha_{\mathcal{S}}) = \alpha_{\mathcal{T}}$; i.e., r induces a function $r^* : \mathcal{A}[\mathcal{S}] \to \mathcal{A}[\mathcal{T}]$.

3. For all $\alpha_{\mathcal{T}}, \alpha'_{\mathcal{T}} \in \mathcal{A}[\mathcal{T}]$, it holds that $\alpha_{\mathcal{T}} \to_{\mathcal{T}} \alpha'_{\mathcal{T}}$ if and only if there exist $\alpha_{\mathcal{S}}, \alpha'_{\mathcal{S}} \in \mathcal{A}[\mathcal{S}]$ such that $r^*(\alpha_{\mathcal{S}}) = \alpha_{\mathcal{T}}$, $r^*(\alpha'_{\mathcal{S}}) = \alpha'_{\mathcal{T}}$ (in the sense of condition (2)), and $\alpha_{\mathcal{S}} \to_{\mathcal{S}} \alpha'_{\mathcal{S}}$. That is, every valid assembly sequence of $\mathcal{T}$ can be "mimicked" by $\mathcal{S}$, but no other assembly sequences can be so mimicked, so that the meaning of the relation $\to$ is preserved by r^*. [2]

Let $\mathfrak{C}$ be a class of singly-seeded tile assembly systems, and let U be a tile set (with tile assembly systems having tile set U not necessarily elements of $\mathfrak{C}$). Note that every element of $\mathfrak{C}$, REPR, and $\mathsf{FIN}(U)$ is a finite object, hence can be represented in a suitable format for computation in some formal system such as Turing machines. We say U is *(intrinsically) universal* for $\mathfrak{C}$ if there are computable functions $R : \mathfrak{C} \longrightarrow \mathsf{REPR}$ and $A : \mathfrak{C} \longrightarrow \mathsf{FIN}(U)$ such that, for each $\mathcal{T} = (T, \sigma_{\mathcal{T}}, \tau) \in \mathfrak{C}$, there is a constant $c \in \mathbb{N}$ such that, letting $r = R(\mathcal{T})$, $\sigma = A(\mathcal{T})$, and $\mathcal{U}_{\mathcal{T}} = (U, \sigma, \tau)$, $\mathcal{U}_{\mathcal{T}}$ simulates $\mathcal{T}$ with resolution loss c and representation function r. That is, $R(\mathcal{T})$ outputs a representation function that interprets assemblies of $\mathcal{U}_{\mathcal{T}}$ as assemblies of $\mathcal{T}$, and $A(\mathcal{T})$ outputs the seed assembly used to program tiles from U to represent the seed tile of $\mathcal{T}$.

[2] Here is another fundamental difference between Ollinger's notion of a global transition function, and our notion of "transitioning" from one assembly to a larger assembly. Ollinger defines "rescaling" of a cellular automaton in such a way that it makes sense to talk about a global transition function that, in a single step, simulates multiple time steps of the simulating cellular automaton (enough of them to simulate a single time step of the simulated cellular automaton). The inherent nondeterminism of tile assembly, and in particular the choice of which frontier location should receive a tile in the next time step, makes such a related definition awkward and perhaps counterproductive, as the number of time steps before a new block is completed is variable, and in particular may be much more than the number of tiles that must be placed in the block before it is completed. Therefore, we keep the original "timescale" of the simulating TAS, rather than speed it up in an attempt to synchronize the time steps of the simulated and simulating TAS's.

www.manaraa.com

5.3 An Intrinsically Universal Tile Set

In this chapter, we exhibit an intrinsically universal tile set for any "nice" tile assembly system. Before proceeding, we must first define the notion of a "nice" tile assembly system. Let $\mathcal{T} = (T, \sigma, 2)$ be a tile assembly system, and $\vec{\alpha}$ be an assembly sequence in $\mathcal{T}$ whose result is denoted as α. We say that $\mathcal{T}$ is *locally consistent* if the following conditions hold.

1. For all $\vec{m} \in \text{dom } \alpha - \text{dom } \sigma$,

$$\sum_{\vec{u} \in \text{IN}^{\vec{\alpha}}(\vec{m})} \text{str}_{\alpha(\vec{m})}(\vec{u}) = 2,$$

   where $\text{IN}^{\vec{\alpha}}(\vec{m})$ is the set of sides on which the tile that $\vec{\alpha}$ places at location $\vec{m}$ initially binds. That is, every tile initially binds to the assembly with exactly bond strength equal to 2 (either a single strength 2 bond or two strength 1 bonds).

2. For all producible assemblies $\alpha \in \mathcal{A}[\mathcal{T}]$, $\vec{u} \in U_2$, and $\vec{m} \in \text{dom } \alpha$, if $\alpha(\vec{m} + \vec{u})$ is defined, then the following condition holds:

$$\text{str}_{\alpha(\vec{m})}(\vec{u}) > 0 \Rightarrow \text{label}_{\alpha(\vec{m})}(\vec{u}) = \text{label}_{\alpha(\vec{m}+\vec{u})}(-\vec{u}) \text{ and } \text{str}_{\alpha(\vec{m})}(\vec{u}) = \text{str}_{\alpha(\vec{m}+\vec{u})}(-\vec{u}).$$

While condition (1) of the above definition is reminiscent of the first condition of local determinism [54], the second condition says that there are no (positive strength) label mismatches between abutting tiles. However, we must emphasize that a locally consistent tile assembly system need not be directed, and moreover, even a locally deterministic tile assembly system need not be locally consistent because of the lack of any kind of "determinism restriction" in the latter definition. The main result of this chapter is the following.

Theorem 26. Let $\mathfrak{C}$ be the set of all locally consistent tile assembly systems. There exists a finite tile set U that is intrinsically universal for $\mathfrak{C}$.

In the remainder of this chapter, we prove Theorem 26, that is, we show that for every locally consistent tile assembly system $\mathcal{T} = (T, \sigma, 2)$, there exists a seed assembly $\sigma_{\mathcal{T}}$, such that the tile assembly system $\mathcal{U}_{\mathcal{T}} = (U, \sigma_{\mathcal{T}}, 2)$ simulates $\mathcal{T}$ with a resolution loss $c \in \mathbb{N}$ that depends only on the glue complexity of $\mathcal{T}$. Instead of giving an explicit (and tedious) definition of the tile types in U, we implicitly define U by describing how $\mathcal{U}_{\mathcal{T}}$ simulates $\mathcal{T}$.

www.manaraa.com

5.3.1 High-Level Overview

Intuitively, $\mathcal{U}$ simulates $\mathcal{T}$ by growing "supertiles" that correspond to tile types in T. In other words, every supertile is a $c \times c$ block of tiles that is mapped to a tile type $t \in T$. To do this, each supertile that assembles in $\mathcal{U}_{\mathcal{T}}$ contains the full specification of T as a lookup table (a long row of tiles that encodes all of the information in the set of tile types T), analogous to the genome of an organism being fully replicated in each cell of that organism, no matter how specialized the function of the cell. This lookup table is carefully propagated through each supertile in $\mathcal{U}_{\mathcal{T}}$ via a series of "rotation" and "copy" operations – both of which are well-known self-assembly primitives.

In the table, we represent each (glue,direction) pair as a binary string, and represent the tile set as a table mapping 1-2 input glue(s) to 0-3 output glue(s). Since each tile type of $\mathcal{T}$ may not have well-defined input sides, when two supertiles representing tiles of $\mathcal{T}$ must potentially cooperate to place a new supertile within a block adjacent to both of them, it is imperative that each grows into the block in such a way as to remain unobtrusive to the other supertile. This is done with a "probe" that grows toward the center of the block, as shown in Figure 5.1. At the moment the probes meet in the middle, they "find out" in what direction the other input supertile lies, and at that point decide in which direction to grow the rest of the forming supertile. so as to avoid the tiles that were already placed as part of the probes. We do not know how to deal with three probes at once, which is the reason both parts of the definition of locally consistent, which imply that only two input probes will ever be present at one time. The next step is to bring the values of two input glues together before doing a lookup on the table, because they are both needed to simulate cooperation. The table must be read and copied at the same time, otherwise the planarity of the tiles would hide the table as it is read and it could not be propagated to the output supertiles. Many choices made in the construction, such as the relative positioning of glues/table, or the counter-clockwise order of assembly, are choices that simply were convenient and seemed to work, but are not necessarily required.

www.manaraa.com

5.3.2 Construction of the Lookup Table

In order to simulate the behavior of $\mathcal{T}$ with $\mathcal{U}_{\mathcal{T}}$, we must first encode the definition of T using tiles from U. We will do this by constructing a "glue lookup table," denoted as $\mathbf{T}_{\mathcal{T}}$, and is essentially the self-assembly version of a kind of hash table. Informally, $\mathbf{T}_{\mathcal{T}}$ is a (very) long string (of tiles from U) consisting of two copies of the definition of the tile set T separated by a small group of spacer symbols. The left copy of the lookup table is the *reverse* of the right copy. The lookup table maps all possible sets of input sides for each tile type $t \in T$ to the corresponding sets of output sides.

5.3.2.1 Addresses

The lookup table $\mathbf{T}_{\mathcal{T}}$ consists of a contiguous sequence of "addresses," which are formed from the definition of T. Namely, for each tile type $t \in T$, we create a unique binary key for each combination of sides of t whose glue strengths sum to exactly 2. Each of these combinations represents a set of sides which *could* potentially serve as the input sides for a tile of type t in a producible assembly in $\mathcal{T}$.

We say that a *pad* is an ordered triple (g, d, s) where g is a glue label in T, $d \in \{N, S, E, W\}$ is an edge direction, and $s \in \{0, 1, 2\}$ is an allowable glue strength. Note that a set of four pads – one for each direction d – fully specifies a tile type. We use $\text{Pad}(t, d)$ to denote the pad on side d of the tile type $t \in T$

Let $\text{Bin}(p)$ be the binary encoding of a pad $p = (g, d, s)$, consisting of the concatenation of the following component binary strings:

1. g (*glue specification*): Let G be the set of glue types from all edges with positive glue strengths in $T \cup \{g_{\text{null}}\}$ (a.k.a., the null glue). Fix some ordering $g_{\text{null}} \leq g_0 \leq g_1 \leq \cdots$ of the set G. The binary representation of g_i is the binary value of i padded with 0's to the left (as necessary) to ensure that the string is exactly $\lceil \log(|G| + 1) \rceil$ bits.

2. d (*direction*): If $d = N$ (E, S, or W), append 00 (01, 10, or 11, respectively).

3. s (*strength*): If $s = 1$ (2) append 0 (1).

www.manaraa.com

Note that $\lceil \log(|G| + 1) \rceil + 2 + 1$ is the length of the binary string encoding an arbitrary pad p, and is a constant that depends only on T.

An *address* is a binary string that represents a set of pads which, themselves, can potentially serve as the input sides of some tile type $t \in T$. It can be composed of one of the two following binary strings:

1. A prefix of zeros, $0^{\lceil \log(|G|+1) \rceil + 3}$, followed by $\text{Bin}(p)$ for $p = (g, d, 2)$, or
2. the concatenation of $\text{Bin}(p_1)$ and $\text{Bin}(p_2)$ for $p_1 = (g_1, d_1, 1)$ and $p_2 = (g_2, d_2, 1)$. The ordering of $\text{Bin}(p_1)$ and $\text{Bin}(p_2)$ in an address must be consistent with the following orderings: EN, SE, WS, NW, NS, EW.

Note that it is possible for more than one tile type $t \in T$ to share a set of input pads and therefore an address.

5.3.2.2 Encoding of T

We will now construct the string $w_{\mathcal{T}}$, which will represent the definition of T. Intuitively, $w_{\mathcal{T}}$ will be composed of a series of "entries." Each entry is associated to exactly one address of a tile type $t \in T$ and specifies the pads for the output sides of t. In this way, once the input sides for a supertile have formed, the corresponding pads can be used to form an address specifying (a set of) appropriate output pads. Note that since more than one tile type may share an address in a nondeterministic tile assembly system, more than one tile type may share a single entry.

We define an *entry* to be a string beginning with '#' followed by zero or more "sub-entries", each corresponding to a different tile type, separated by semicolons. Let A be the set of all binary strings representing every address created for each $t \in T$. The string $w_{\mathcal{T}}$ will consist of $1 + \max A$ entries for addresses 0 to $\max A$. The i^{th} entry, denoted as e_i, corresponds to the i^{th} address, which may or may not be in A (if it is not, then e_i is empty).

We say that a *sub-entry* consists of a string specifying the pads for the output sides of a tile type $t \in T$. Let e_i be the entry containing a given sub-entry (note that i is the address of e_i),

www.manaraa.com

and $T_i \subseteq T$ be the set of tile types addressable by i (i.e., the set of tile types for which i is a valid address). The entry e_i will be comprised of exactly $|T_i|$ sub-entries. For $0 \leq k < j$, the k^{th} sub-entry in e_i, where $t_k \in T_i$ is the k^{th} element of T_i (relative to some fixed ordering), is the string $\text{OUT}(N), \text{OUT}(E), \text{OUT}(S), \text{OUT}(W)$ (the commas in the previous string are literal) with $\text{OUT}(d) = \text{Bin}(\text{Pad}(t_k, d))^R$ if the glue for $\text{Pad}(t_k, d)$ is not g_{null} and d is not a component of the address i, otherwise $\text{OUT}(d) = \lambda$. Intuitively, a sub-entry is a comma-separated list of the (reversed) binary representations of the pads for an addressed tile type, but including only pads whose glues are not g_{null} and whose directions are not a part of the address (and therefore input sides). We will now use the string $w_{\mathcal{T}}$ to construct the lookup table $\mathbf{T}_{\mathcal{T}}$.

5.3.2.3 Full specification of $\mathbf{T}_T$

We now give the full specification for the lookup table $\mathbf{T}_{\mathcal{T}}$. First, define the following strings: $w_0 =$ '>', $w_1 =$ '< %% >', $w_2 =$ '<'. Now let $\mathbf{T}_{\mathcal{T}}$ be as follows: $\mathbf{T}_{\mathcal{T}} = \text{sb}(w_0 \circ w_{\mathcal{T}} \circ w_1 \circ (w_{\mathcal{T}})^R \circ w_2)$, where, for strings x and y, $x \circ y$ is the concatenation of x and y, and $\text{sb} : \Sigma^* \to \Sigma^*$ is defined to "splice blanks" into its input: between every pair of adjacent symbols in the string x, a single '␣' (blank) symbol is inserted to create $\text{sb}(x)$. This splicing of blanks is required to be able to read from the table without "locking it from view", when reading the table for operations that require growing a column of tiles in towards the table (as opposed to away from it), a blank column is used, and for growing a column away from the table, a symbol column is used so that the symbol can be propagated to the top of the column for later copying.

5.3.2.4 The Lookup Procedure

In our construction, when a supertile t^* that is simulating a tile type $t \in T$ forms, we must overcome the following problem: once we combine the input pads (given as the output pads of the supertiles to which t^* attaches), how do we use $\mathbf{T}_{\mathcal{T}}$ to lookup the output pads for t^*? In what follows, we briefly describe how we achieve this. In other words, we show how an address, a string of random bits, and a copy of $\mathbf{T}_{\mathcal{T}}$ are used to compute the pad values for the

www.manaraa.com

non-input sides of a supertile. A detailed figure and example of this procedure can be found in the technical appendix.

For ease of discussion and without loss of generality, we assume that the row of tiles encoding $\mathbf{T}_{\mathcal{T}}$ (assembled West to East) and the column of tiles encoding an address and a random string of bits (assembled North to South at the West end of $\mathbf{T}_{\mathcal{T}}$) are fully assembled, forming an 'L' shape with no tiles in the area between them. For other orientations of the table and address the logical behavior is identical, simply rotated.

Intuitively, the assembly of the lookup procedure assembles column wise in a zig-zag fashion from left to the right. In the "first phase," a counter initialized to 0 is incremented in each column where the value of the tile in the representation of $\mathbf{T}_{\mathcal{T}}$ is a ';', thus counting up at each entry contained in $\mathbf{T}_{\mathcal{T}}$. Once that number matches the value of the given address (which, along with the random bits is copied through this procedure), the entry e corresponding to that address has been reached and a new counter begins which counts the number of sub-entries n in that entry. Note that for directed tile systems, $n \leq 1$. Once the end of that entry is encountered, yet another counter, initialized to 0, begins and increments on each remaining entry until the end of the first copy of $w_{\mathcal{T}}$ is reached (the number n is propagated to the right). This counts the number of entries, denoted as m, between e and the end of the lookup table. The "second phase" is used to perform, in some sense, an operation equivalent to calculating $p = b \mod n$, where b is the binary value of the string of random bits required for the lookup procedure (this is how we simulate nondeterministic assemblies). This selects the index of the sub-entry in e which will be used, completing the random selection of one of the possibly many tile types contained in entry e.

In the current version of our construction, we merely use a random number selection procedure reminiscent of the more involved (but more uniform) random selection procedures discussed in [19]. Although it is possible to incorporate these more advanced techniques into our construction (and thus achieve a higher degree of uniformity in the simulation of randomized tile systems), we choose not to do so for the sake of simplicity.

Next, a reverse counter, a.k.a., a subtractor, counts down at each entry from m to 0, and

www.manaraa.com

by the way we constructed $\mathbf{T}_{\mathcal{T}}$, this final counter obtains the value 0 at the entry e (in the reverse of $w_{\mathcal{T}}$). Now, another subtractor counts from p to 0 to locate the correct sub-entry that was selected randomly. Finally, each pad in the sub-entry is rotated "up and to the right," and the group of pads is propagated through the remainder of the lookup table, thus ending with the values of the non-input pads represented in the rightmost column.

5.3.3 Supertile Design

A supertile s is a subassembly in $\mathcal{U}_{\mathcal{T}}$ consisting of a $c \times c$ block of tiles from T, where c depends on the glue complexity of T. Each s can be mapped to a unique tile type $t \in T$. In our construction there are two logical supertile designs. The first, denoted *type-0*, simulates tile additions in $\mathcal{T}$ in which there are 2 input sides, each with glue strength = 1. The second, denoted *type-1*, simulates the addition of tiles via a single strength 2 bond.

While there are several differences in the designs of *type-0* and *type-1* supertiles, one commonality is how their edges are defined. Namely each input or output edge of any supertile is defined by the same sequence of variable values. Since the edges for each direction are rotations of each other, we will discuss only the layout of the south side of a supertile. From left to right, the tiles along the south edge of a supertile will represent a string formed by the concatenation (in order) of the strings: $\mathbf{T}_{\mathcal{T}}$, $\text{Bin}(\text{Pad}(t, S))$, $0^{c'}$, $\text{Bin}(\text{Pad}(t, S))$, and $\mathbf{T}_{\mathcal{T}}$. Note that c' is a constant that depends on the glue complexity of T.

5.3.3.1 Type-0 Supertiles (i.e., simulating tiles that attach via two single-strength bonds)

When a tile binds to an assembly in $\mathcal{T}$ with two input sides whose glues are each single strength, there are $\binom{4}{2} = 6$ possible combinations of directions for those input sides: north and east (NE), north and south (NS), north and west (NW), east and south (ES), east and west (EW), and south and west (SW). These combinations can be divided into two categories, those in which the sides are opposite each other (NS and EW), and those in which the sides are adjacent to each other (NE, NW, ES, and SW).

www.manaraa.com

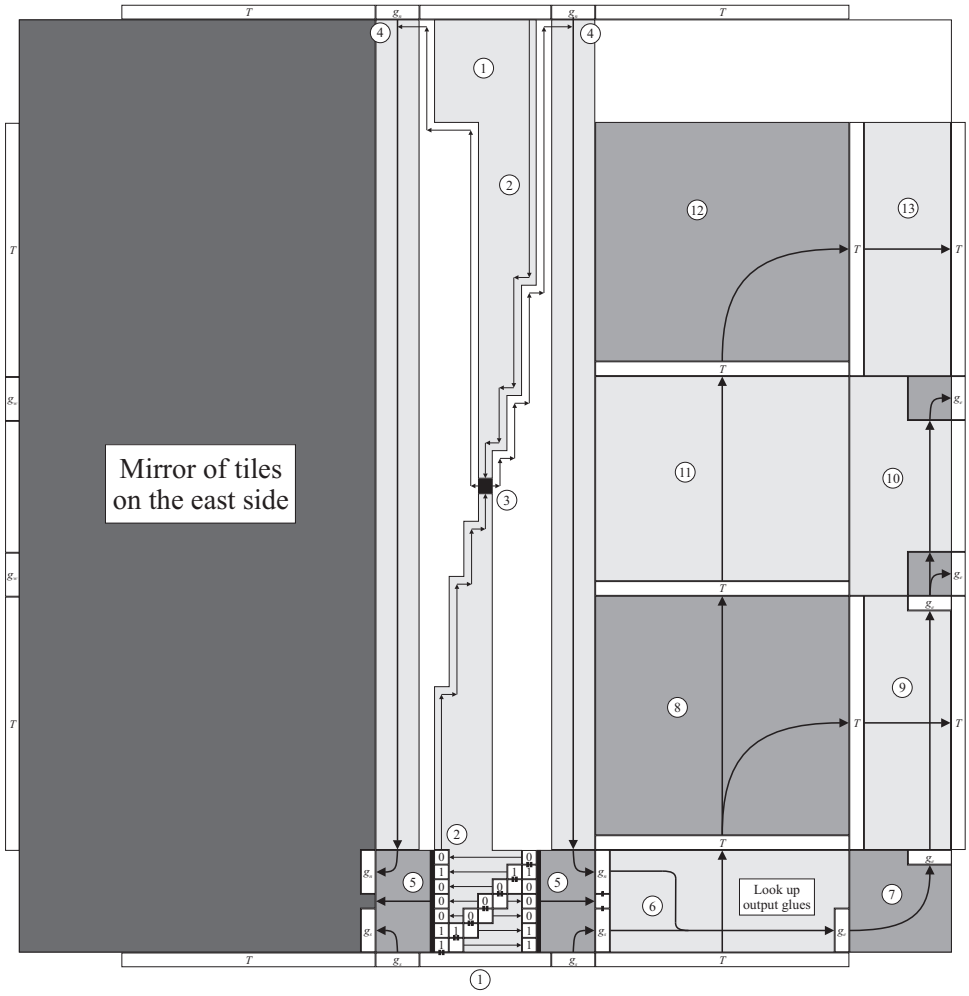


Figure 5.1: NS supertile

www.manaraa.com

Opposite Input Sides: Supertiles which represent tile additions with two opposite input sides, NS and EW, are logically identical to rotations of each other, so here we will only describe the details of a supertile with NS input sides. Figure 5.1 shows a detailed image depicting the

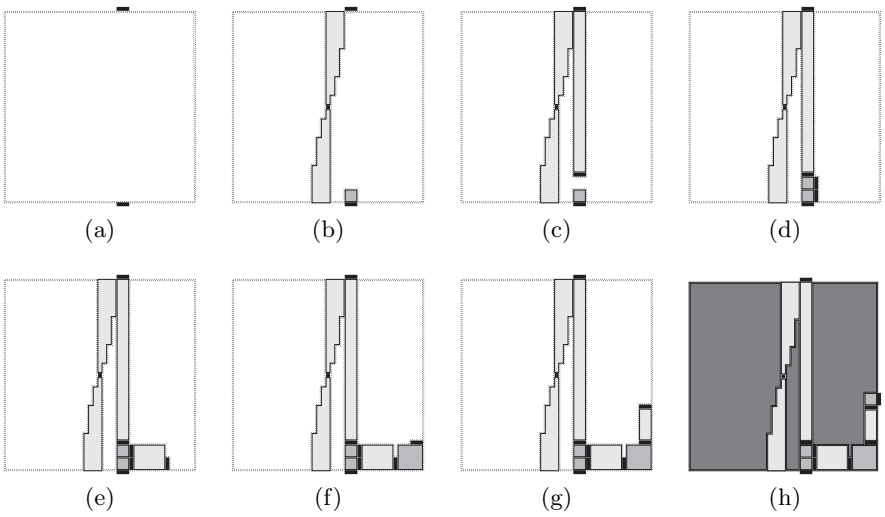


Figure 5.2: Intuitive depiction of (a portion of) the self-assembly of a type-0 supertile. Note that the lookup procedure is performed in (d) and (e).

formation of an NS supertile, with arrows giving the direction of growth for each portion and numbers specifying the order of growth. For ease of discussion and without loss of generality, we assume that the rows of tiles which form the input sides of a supertile have fully formed before any other part of the supertile assembles. The first portions to assemble are the center blocks to the interior of each input side, labeled 1. This subassembly forms a square in which a series of nondeterministic selections of tile types is used to generate a random sequence of bits. These bits are propagated to the left and right sides of the block, to ensure that each side uses the same random bits for the randomized selection after the sides have been "sealed off" from each other by "probes" described next. Once that block has completed, a log-width binary subtractor, which is half the width of the block, assembles. The subtractors from the north and south count down from a specified value (that depends on $\mathcal{T}$ and is encoded into the seed supertile) to 0, and shrink in width until they terminate at positions adjacent to the center square of the block. These subtractors are "probes" that grow to the center where the

www.manaraa.com

direction of the input sides (the *type*) is detected. It is at this point that the central (black in the figure) tile can attach. It is this tile which determines the *type* of the supertile (NS in this case) because it is unique to the combination of directions from which the inputs came. At this point, symmetry is broken and two paths of tiles assemble from the center back towards the north side. They in turn initiate the growth of subassemblies which propagate the value of the north input pad down towards the South of the supertile. Once that growth nears the southern side, the two input pads are rotated and brought together, with this combination of input pads forming an *address* in the lookup table. In the manner described previously, this address along with the random bits generated within block 1 (which are also passed through block 5) is used to form the subassembly of block 6 whose southern row contains a representation of $\mathbf{T}_{\mathcal{T}}$ and results in the correct output pads being represented in the final column of that block. Note that Figure 5.1 only shows the details of the east side of the block since the West side is an identical but rotated version. Finally, subassemblies 7 through 13 form which rotate and pass the necessary information to the locations where it must be correctly deposited to form the output sides of the supertile. Every side of a supertile that is not an input side receives an output pad, even if it is for the null glue (in which case it does not initiate the growth of the input side of a possible adjacent supertile).

5.3.4 Lookup Example

We now consider an example tile assembly system $\mathcal{T} = (T, \sigma, 2)$ where T is the tile set defined in Figure 5.3 and σ is simply a single tile of type S located at point $(0, 0)$. In this section we will describe how $\mathcal{T}$ can be simulated using T_{univ}. The set of glues, G, for T is (a, b, c, d), with binary representations $(001, 010, 011, 100)$, respectively, and 000 reserved for g_{null}.

The value for a_{max} will be the address for tile type I corresponding to '$(d, S, 1), (c, E, 1)$', namely 100100011010.

www.manaraa.com

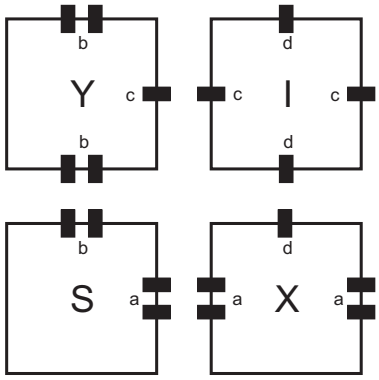


Figure 5.3: Example tile set T_{sim}

Table 5.1: Pads and their binary representations (Note that there are no spaces in the actual binary representations, they've just been added to this table for convenience)

Pad	Bin(Pad)
$(b, N, 2)$	010 00 1
$(a, E, 2)$	001 01 1
$(g_{null}, S, 0)$	000 10 0
$(g_{null}, W, 0)$	000 11 0
$(c, E, 1)$	011 01 0
$(b, S, 2)$	010 10 1
$(a, W, 2)$	001 11 1
$(d, N, 1)$	100 00 0
$(d, S, 1)$	100 10 0
$(c, W, 1)$	011 11 0

5.3.5 Additional Details

Adjacent Input Sides: Supertiles which represent tile additions with two adjacent input sides, NE, NW, ES, and SW, are logically identical to rotations of each other, so here we will only describe the details of a supertile with SW input sides. Figure 5.5 shows a detailed image depicting the formation of an SW supertile. It is logically very similar to an NS supertile. However, once the center, black tile is placed then a row grows to the west, which initiates the growth of the subassemblies that propagate the W input pad to the proper location to be combined with the S pad. After the output pads are determined, then both the north and the

www.manaraa.com

Table 5.2: Tile types, their possible addresses, and the binary representations of those addresses

Tile type	Addresses	Binary representations
S	$(b, N, 2)$	000 00 0 010 00 1
	$(a, E, 2)$	000 00 0 001 01 1
X	$(a, E, 2)$	000 00 0 001 01 1
	$(a, W, 2)$	000 00 0 001 11 1
Y	$(b, S, 2)$	000 00 0 010 10 1
	$(b, N, 2)$	000 00 0 010 00 1
I	$(c, E, 1), (d, N, 1)$	011 01 0 100 00 0
	$(d, N, 1), (d, S, 1)$	100 00 0 100 10 0
	$(d, N, 1), (c, W, 1)$	100 00 0 011 11 0
	$(d, S, 1), (c, E, 1)$	100 10 0 011 01 0
	$(c, E, 1), (c, W, 1)$	011 01 0 011 11 0
	$(c, W, 1), (d, S, 1)$	011 11 0 100 10 0

east sides are assembled using the information from that single lookup table.

5.3.5.1 Type-1 Supertiles: simulating tiles that attach via a single double-strength bond)

Supertiles that represent tile additions with a single, strength 2, input side (N, S, E, or W) are logically identical to rotations of each other, so here we will only describe the details of a supertile with a S strength 2 input side. Figure 5.6 shows a detailed image depicting the formation of an strength 2, S supertile. Notice that for supertiles of this type, the address for the lookup table is formed from only one pad, so as soon as the assembly of block 1 completes with the random string of bits, then the pad can be rotated up and the output pads computed. Finally, all three output pads are propagated around and "dropped off" in their respective locations.

5.3.6 Constructing the Seed Supertile: $\sigma_{\mathcal{T}}$

Here we describe how the seed $\sigma_{\mathcal{T}}$ for $\mathcal{U}_{\mathcal{T}}$ is formed. First, using the procedure described previously, the lookup table $\mathbf{T}_T$ is generated. Then, using the value of $|\mathbf{T}_T|$ and the lengths of the binary representations of pads in T, the value to be used as the maximum value for

www.manaraa.com

the binary subtractors that grow to the center of type-0 supertiles can be calculated. Finally, tile types unique to the seed assembly are used to fill in the corners and interior of the the supertile that represents $\sigma_{\mathcal{T}}$.

5.4 Conclusion

We have shown that there is a single tile set U in Winfree's abstract tile assembly model, which can be "programmed" through appropriate setting of a finite seed assembly, to simulate a scaled version of the assembly process of any in a wide class of *locally consistent* tile assembly systems, those that have the properties that each tile binds with exactly strength 2, and there are no glue mismatches in any producible assembly. This captures a wide class of tile assembly systems, including counters, square-builders and other shape-building tile assembly systems, and the tile assembly systems described in [2, 31, 47, 54].

5.4.1 A Universal Computer Simulating Itself

We note that all the tile assembly systems produced using tiles from U by our construction satisfy the two properties of local consistency. We assumed for simplicity that the simulated system $\mathcal{T}$ is singly-seeded, whereas clearly $\mathcal{U}$ is not since the setting of a complex seed assembly is the means by which U is programmed. If we wish U to be able to simulate systems with larger seeds, say, k tiles in size, this can of course be done by programming the seed assembly of U to start with k blocks instead of a single block. But this does not imply U can simulate itself, as this construction can only simulate tile assembly systems with a strictly smaller seed assembly, if the resolution loss is 2 or greater.

The most theoretically striking reason to require a "universal computer" to be contained within the class of computers that it simulates is the idea that a computer can simulate itself. This result is used, with deep results, in the recursion theorem, and in Peter Gács' construction [24] of a fault-tolerant one-dimensional cellular automaton. While U (or more accurately tile assembly systems with U as the tile set) is in some sense contained in the class of tile assembly systems that it simulates, it cannot simulate itself simulating itself simulating

www.manaraa.com

itself ... etc., as in Gács' construction. At some finite level, we must have "$\mathcal{U}$ simulates $\mathcal{U}$ simulating $\mathcal{U}$ ... simulating $\mathcal{U}$ simulating some other system $\mathcal{T}$".

5.4.2 Resolution Loss

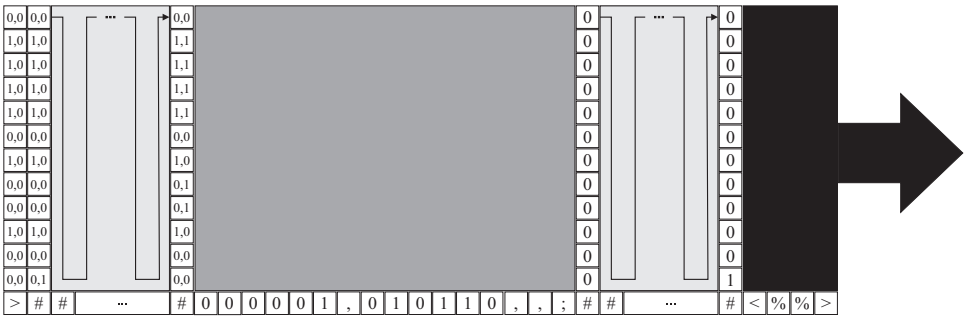
We omit a formal analysis, but it is easy to bound the resolution loss of our construction in terms of the number of glues g in the system. In particular, the table used for lookup contains one entry for each pair of strength-1 glues in the system, and 1 entry for each strength-2 glue, and each entry consists of at most three output glues, each of which requires $\log g$ bits to represent. The size of this table asymptotically dominates the size of the blocks, so the resolution loss c is at most $O(g^2 \log g)$.

This is not tight for certain tile assembly systems. In particular, let G be the $g \times g$ Boolean matrix where entry (i, j) is 1 if the i^{th} and j^{th} glues ever function as inputs together[3], and 0 otherwise. If G is dense, then our resolution loss bound is tight, and if G is sparse, then it is not tight. It is not difficult to construct tile assembly systems to achieve either condition, implying in particular that our encoding of the table is inefficient for certain tile assembly systems.

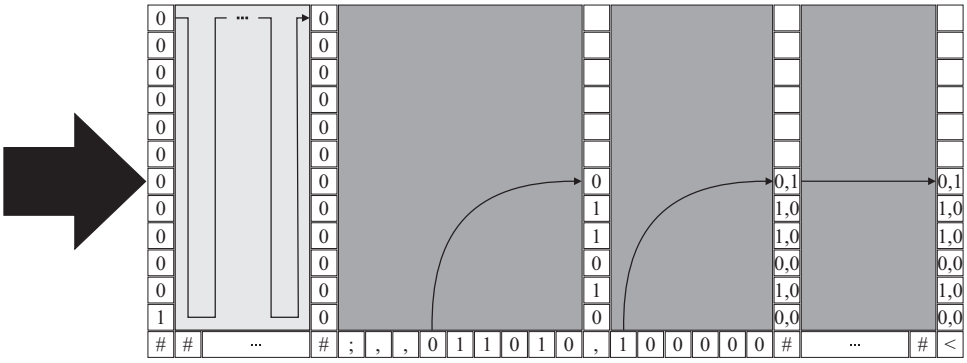
An alternative encoding, which is more complicated to implement, is to store the table as key-value pairs, in which the input glues and output glues are encoded explicitly in each entry of the table, the input glues (addresses) serving as keys and the output glues serving as values. The tiles doing the lookup would then do a comparison between the address being searched and the keys stored in the table, rather than counting to the correct entry using the address as the start value for the counter. By storing only entries in which the addresses are actually used in some assembly sequence, the size of the table can be made $O(g' \log g)$, where g' is the number of pairs of input glues that are actually used. Since each glue that is an input for one tile type is an output for another, this implies $g' \geq g/2 = \Omega(g)$. Hence the savings is potentially up to a multiplicative factor of g.

[3]More precisely, if there exists an assembly sequence in which a tile type having those glues ever binds with positive strength on each of the glues

www.manaraa.com



(a) The initial phase of the lookup procedure. Notice the pairs of bits stored in the leftmost column. The leftmost bits represent the address of the desired set of output glues; the rightmost bits represent the counter used to "search" for appropriate entry in the table. The blank region is where we "randomly" select the output glues.



(b) The final phase of the lookup procedure. The subentries corresponding to the output glues are rotated up into column format and pass through the remainder of the table.

Figure 5.4: Example lookup of the output pads for address 011110100100 (or "$(c, W, 1), (d, S, 1)$"). Assume the sub-assembly begins with the bottom and left rows completely formed, and then proceeds column by column from left to right.

www.manaraa.com

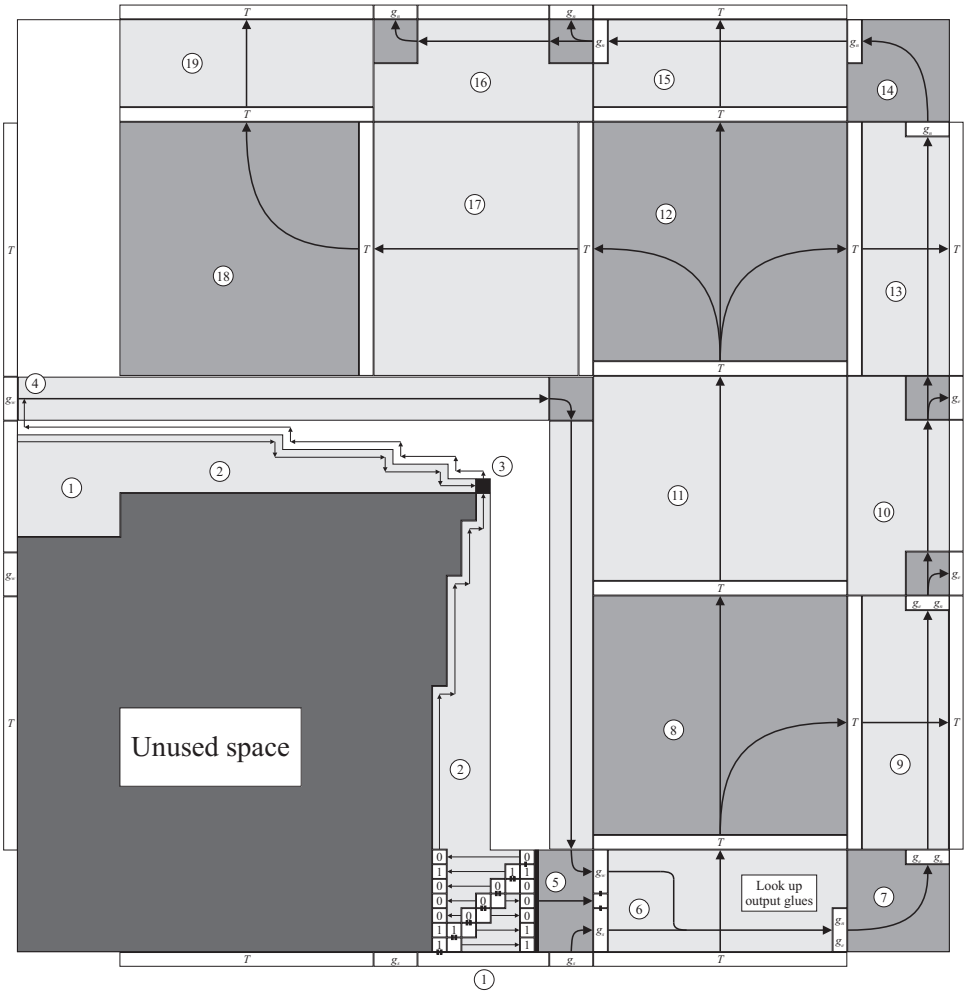


Figure 5.5: SW supertile

www.manaraa.com

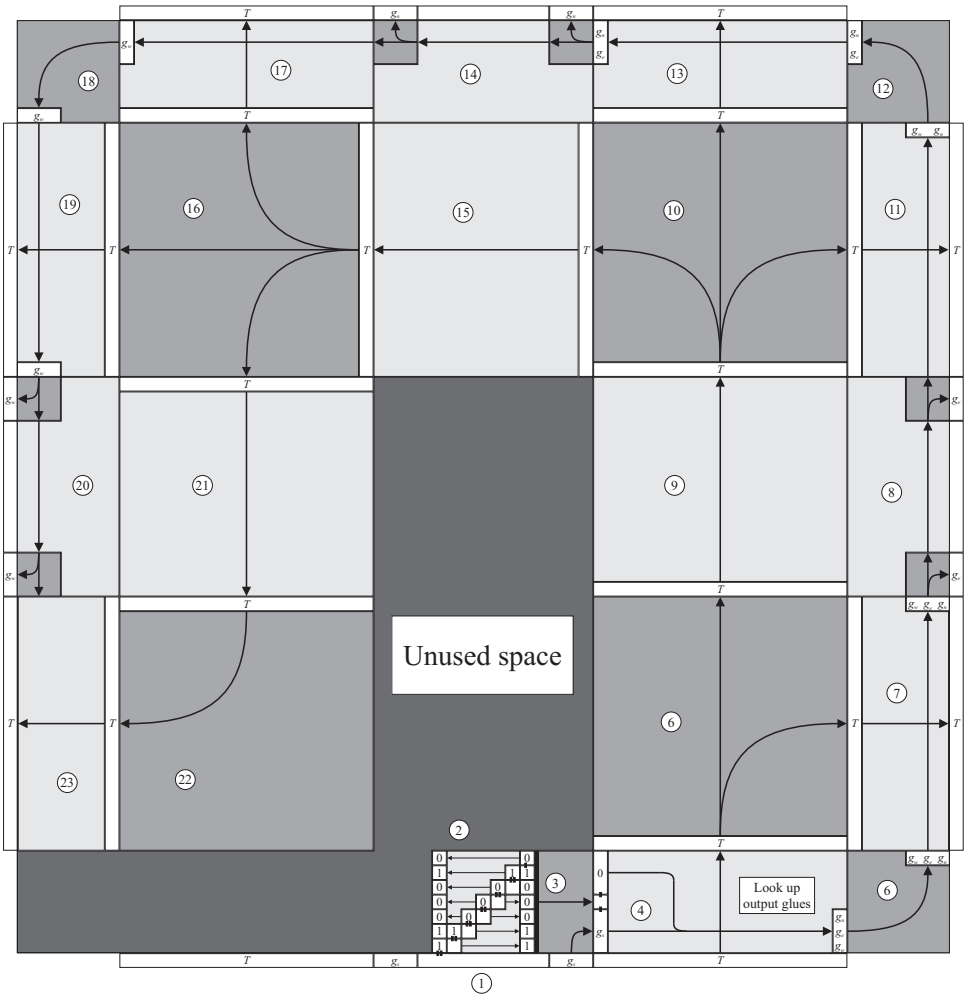


Figure 5.6: Strength 2 supertile

www.manaraa.com

CHAPTER 6. Toward Fault-Tolerant Universality in the Fuzzy Temperature Model

This chapter is joint work with David Doty, Matthew J. Patitz, Dustin Reishus and Robert T. Schweller.

6.1 Introduction

Winfree defined two models of tile-based self-assembly, the abstract Tile Assembly Model (aTAM) and the kinetic Tile Assembly Model (kTAM). In both models, the fundamental components are un-rotatable, but translatable square "tile types" whose sides are labeled with glue "colors" and "strengths." Two tiles that are placed next to each other *interact* if the glue colors on their abutting sides match, and in the aTAM, a tile *binds* to an assembly if it interacts on all sides with total strength at least a certain ambient "temperature," usually taken to be 2. In particular, if a tile has two strength-1 glues, both of them must match the corresponding glues in the assembly in order to remain bound.

However, in the more thermodynamically plausible kTAM, tiles may bind even if they interact with strength less than 2, but are assumed to detach at a rate inversely and exponentially proportional to the strength with which they interact. Hence tiles attached with strength 1 detach "quickly", and tiles attached with strength 2 detach "slowly". A tile attached with only strength 1 (a so-called "insufficient attachment") represents a potential error, as its other strength-1 glue may be mismatched with the abutting portion of the assembly, or mismatched with what is eventually intended to be placed at that position. However, since strength-1 attachments are assumed to detach after a short time, an insufficient attachment actualizes into a permanent error only if another tile first binds to secure the faulty tile in place, causing

www.manaraa.com

the entire assembly to become stable at temperature 2. That is, by "wandering" temporarily through the space of assemblies producible at temperature 1, we may arrive at an assembly not producible at temperature 2, yet that, once formed, is stable at temperature 2. The development of physical and algorithmic mechanisms for preventing such errors remains a formidable challenge in nanoscale self-assembly.

Stated informally, the kTAM refines the aTAM by endowing it with a mechanism for error (temporary binding of tiles with strength 1) as well as a mechanism for error correction (eventual detachment of tiles, even those bound with strength 2). Indeed, numerous papers have used these two mechanisms for high-probability error correction in the kTAM [10, 11, 13, 42, 59, 61]. In each of these papers except [59], the same basic principle is used to achieve error correction, known as *proofreading*. If an insufficient attachment results in mismatching glues, this error is "amplified" by forcing further growth to require many other insufficient attachments to stabilize. Since these happen only slowly, the assembly process is slowed down, giving time for the tiles that stabilized the original insufficient attachment to detach, thus correcting the original error. In [59], Winfree also shows how errors are removed through the detaching of tiles, although there is no "error-amplification process"; Winfree shows that by setting the ratio of the forward rate to the reverse rate sufficiently small (thus slowing down the entire assembly process), erroneous tiles will detach with high probability. Other papers have investigated algorithmic correction of other types of errors [12, 49, 50, 52, 60] and physical, rather than algorithmic, mechanisms for error suppression [9, 23, 35].

In this chapter, we work in a variant of the aTAM known as the *two-handed* aTAM [14, 16]. Winfree's original model, the *seeded* aTAM, stipulates that assembly begins from a specially-designated "seed" tile type, and all binding events consist of the attachment of a single tile to the growing assembly that contains the seed. The seed thus serves as a *nucleation point* from which all further growth occurs. In reality, such single-point nucleation is difficult to enforce [49, 50] as tiles with matching glues may attach to each other in solution, even if neither of them is connected to the seed tile. The two-handed aTAM models this sort of growth by dispensing with the idea of a seed, and simply defining an assembly to be producible if 1) it

www.manaraa.com

consists of a single tile (base case), or 2) it results from the stable aggregation of two producible assemblies (recursive case).

Not only is the two-handed aTAM a more realistic model, it allows us to use the geometry of partially-formed assemblies, rather than relying solely on (error-prone) glue specificity, to enforce binding rules between subassemblies. In doing do, we are able to achieve a much stronger notion of fault-tolerance than that described in previous error-correction papers. Informally, our model of fault-tolerance, which we term the *fuzzy temperature* model, is as follows (a formal description is given in Section 6.4). Similarly to the kTAM model, we allow strength-1 insufficient attachments to occur. However, we do not model forward or reverse rates of growth as in the kTAM, as there is no need: any insufficient attachment(s) that lead to an assembly that is stable at temperature 2 *were never errors in the first place*, as such an assembly can always lead to an assembly that was producible with only strength-2 growth. That is, viewed as a modification of the aTAM, we allow the temperature to be "fuzzy", occasionally drifting from 2 down to 1, which allows strength-1 growth for as long as the temperature remains low. However, once the temperature is raised back to 2, thus dissolving any structure that is stable only at temperature 1, the stable assemblies that are left over are all assemblies that are already producible at temperature 2. Therefore, while insufficient attachments can occur, errors due to insufficient attachments cannot occur, since temperature-2 stabilization of such errors, which our construction prevents, is required for the errors to become permanent.

In an attempt to gain insight into the universality of the fuzzy temperature model, we focus our attention on the problem of assembling an $n \times n$ square, a common benchmark problem for demonstrating the use of self-assembly techniques [2,3,8,17,26,27,47]. In particular, our main result is the construction of a tile set with $O(\log n)$ unique tile types (which is close to the $\Omega(\log n / \log \log n)$ optimal lower bound [47]) that uniquely assembles into an $n \times n$ square in the two-handed aTAM at temperature 2, and that has the fuzzy-temperature fault-tolerance property described above. In keeping with the "wandering" analogy from the beginning of this section, our construction allows arbitrary wandering in the space of assemblies producible at temperature 1, but funnels all such wandering towards a single unique terminal assembly, or

www.manaraa.com

towards the oblivion of destruction at temperature 2.[1]

6.2 Two-Handed Abstract Tile Assembly Model

In this section we formally define a variant of Erik Winfree's abstract Tile Assembly Model that models unseeded growth, known as the *two-handed* aTAM [14, 16]. In the two-handed aTAM, any two assemblies can attach to each other, rather than enforcing that tiles can only accrete one at a time to an existing seed assembly. In this section we define the model to allow for infinite assemblies and systems that produce more than one terminal assembly, even though our main construction does not have these properties. We also allow for systems that start with a finite number of tile types, even though the description of our main construction is in terms of an infinite supply of tiles.

$\lg(x)$ and $\log(x)$ each denote the base-2 logarithm of x, and $\ln(x)$ denotes the base-e logarithm of x.

In the subsequent definitions, given two partial functions f, g, we write $f(x) = g(x)$ if f and g are both defined and equal on x, or if f and g are both undefined on x.

Throughout this section, fix a finite set T of tile types. An *assembly* is a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$ defined on at least one input, 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. We write $|\alpha|$ to denote $|\text{dom } \alpha|$, and we say α is *finite* if $|\alpha|$ is finite. For assemblies α and α', we say that α is a *subassembly* 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. Two assemblies α and β are *disjoint* if dom $\alpha \cap$ dom $\beta = \varnothing$. For two assemblies α and β, define the *union* $\alpha \cup \beta$ to be the assembly defined for all $\vec{x} \in \mathbb{Z}^2$ by $(\alpha \cup \beta)(\vec{x}) = \alpha(\vec{x})$ if $\alpha(\vec{x})$ is defined, and $(\alpha \cup \beta)(\vec{x}) = \beta(\vec{x})$ otherwise. Say that this union is *disjoint* if α and β are disjoint.

[1] We emphasize that this is *not* the same as saying that our construction assembles an $n \times n$ square at temperature 1: at temperature 1, many different terminal assemblies can nondeterministically form, most of which are junk. Our construction ensures that when the temperature is raised to 2, all the junk dissolves away, leaving only assemblies that are required to assemble the square, and which could have grown anyway had the temperature remained at 2. In fact, it is an open problem, first stated by Rothemund in [46], to uniquely assemble an $n \times n$ square at temperature 1 using fewer than $2n - 1$ tile types (compared to our use of $O(\log n)$ tile types), and Rothemund conjectured that $2n - 1$ is a strict lower bound for this problem.

www.manaraa.com

The two-handed aTAM [14, 16] allows for two assemblies, both possibly consisting of more than one tile, to attach to each other, in contrast to the seeded aTAM [47, 59] in which one of the attaching objects is assumed to be a single tile, and the other is the assembly containing the unique "seed tile". Since we must allow that the assemblies might require translation before they can bind, we define a *supertile* to be the set of all translations of a τ-stable assembly, and speak of the attachment of supertiles to each other, modeling that the assemblies attach, if possible, after appropriate translation.

Formally, for assemblies $\alpha, \beta : \mathbb{Z}^2 \dashrightarrow T$ and $\vec{u} \in \mathbb{Z}^2$, we write $\alpha + \vec{u}$ to denote the assembly defined for all $\vec{x} \in \mathbb{Z}^2$ by $(\alpha + \vec{u})(\vec{x}) = \alpha(\vec{x} - \vec{u})$, and write $\alpha \simeq \beta$ if there exists $\vec{u}$ such that $\alpha + \vec{u} = \beta$; i.e., if α is a translation of β. Define the *supertile* of α to be the set $\tilde{\alpha} = \{ \beta \mid \alpha \simeq \beta \}$. A supertile $\tilde{\alpha}$ is *τ-stable* (or simply *stable*) if all of the assemblies it contains are τ-stable; equivalently, $\tilde{\alpha}$ is stable if it contains a stable assembly, since translation preserves the property of stability. Note also that the notation $|\tilde{\alpha}| \equiv |\alpha|$ is well-defined, since translation preserves cardinality (and note in particular that even though we define $\tilde{\alpha}$ as a set, $|\tilde{\alpha}|$ does not denote the cardinality of this set, which is always $\aleph_0$).

For two supertiles $\tilde{\alpha}$ and $\tilde{\beta}$, and temperature $\tau \in \mathbb{N}$, define the *combination* set $C^{\tau}_{\tilde{\alpha},\tilde{\beta}}$ to be the set of all supertiles $\tilde{\gamma}$ such that there exist $\alpha \in \tilde{\alpha}$ and $\beta \in \tilde{\beta}$ such that (1) α and β are disjoint, (2) $\gamma \equiv \alpha \cup \beta$ is τ-stable, and (3) $\gamma \in \tilde{\gamma}$. That is, $C^{\tau}_{\tilde{\alpha},\tilde{\beta}}$ is the set of all τ-stable supertiles that can be obtained by attaching $\tilde{\alpha}$ to $\tilde{\beta}$ stably, with $|C^{\tau}_{\tilde{\alpha},\tilde{\beta}}| > 1$ if there is more than one position at which β could attach stably to α.

It is common with seeded assembly to stipulate an infinite number of copies of each tile, but our definition allows for a finite number of tiles as well. Our definition also allows for the growth of infinite assemblies and finite assemblies to be captured by a single definition, similar to the definitions of [32] for seeded assembly.

Given a set of tiles T, define a *state* S of T to be a multiset of supertiles, or equivalently, S is a function mapping supertiles of T to $\mathbb{N} \cup \{\infty\}$, indicating the multiplicity of each supertile in the state. We therefore write $\tilde{\alpha} \in S$ if and only if $S(\tilde{\alpha}) > 0$.

A *(two-handed) tile assembly system* (*TAS*) is an ordered triple $\mathcal{T} = (T, S, \tau)$, where T is

www.manaraa.com

a finite set of tile types, S is the *initial state*, and $\tau \in \mathbb{N}$ is the temperature. Subsequently we assume that $\tau = 2$, unless explicitly stated otherwise. If not stated otherwise, assume that the initial state S is defined $S(\tilde{\alpha}) = \infty$ for all supertiles $\tilde{\alpha}$ such that $|\tilde{\alpha}| = 1$, and $S(\tilde{\beta}) = 0$ for all other supertiles $\tilde{\beta}$. That is, S is the state consisting of a countably infinite number of copies of each individual tile type from T, and no other supertiles. In such a case we write $\mathcal{T} = (T, \tau)$ to indicate that $\mathcal{T}$ uses the default initial state.

Given a TAS $\mathcal{T} = (T, S, \tau)$, define an *assembly sequence* of $\mathcal{T}$ to be a sequence of states $\vec{S} = (S_i \mid 0 \leq i < k)$ (where $k = \infty$ if $\vec{S}$ is an infinite assembly sequence), and S_{i+1} is constrained based on S_i in the following way: There exist supertiles $\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}$ such that (1) $\tilde{\gamma} \in C^{\tau}_{\tilde{\alpha},\tilde{\beta}}$, (2) $S_{i+1}(\tilde{\gamma}) = S_i(\tilde{\gamma}) + 1$,[2] (3) if $\tilde{\alpha} \neq \tilde{\beta}$, then $S_{i+1}(\tilde{\alpha}) = S_i(\tilde{\alpha}) - 1$, $S_{i+1}(\tilde{\beta}) = S_i(\tilde{\beta}) - 1$, otherwise if $\tilde{\alpha} = \tilde{\beta}$, then $S_{i+1}(\tilde{\alpha}) = S_i(\tilde{\alpha}) - 2$, and (4) $S_{i+1}(\tilde{\omega}) = S_i(\tilde{\omega})$ for all $\tilde{\omega} \notin \{\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}\}$. That is, S_{i+1} is obtained from S_i by picking two supertiles from S_i that can attach to each other, and attaching them, thereby decreasing the count of the two reactant supertiles and increasing the count of the product supertile. If $S_0 = S$, we say that $\vec{S}$ is *nascent*.

Given an assembly sequence $\vec{S} = (S_i \mid 0 \leq i < k)$ of $\mathcal{T} = (T, S, \tau)$ and a supertile $\tilde{\gamma} \in S_i$ for some i, define the *predecessors* of $\tilde{\gamma}$ in $\vec{S}$ to be the multiset $\text{pred}_{\vec{S}}(\tilde{\gamma}) = \{\tilde{\alpha}, \tilde{\beta}\}$ if $\tilde{\alpha}, \tilde{\beta} \in S_{i-1}$ and $\tilde{\alpha}$ and $\tilde{\beta}$ attached to create $\tilde{\gamma}$ at step i of the assembly sequence, and define $\text{pred}_{\vec{S}}(\tilde{\gamma}) = \{\tilde{\gamma}\}$ otherwise. Define the *successor* of $\tilde{\gamma}$ in $\vec{S}$ to be $\text{succ}_{\vec{S}}(\tilde{\gamma}) = \tilde{\alpha}$ if $\tilde{\gamma}$ is a predecessor of $\tilde{\alpha}$ in $\vec{S}$, and define $\text{succ}_{\vec{S}}(\tilde{\gamma}) = \tilde{\gamma}$ otherwise. A sequence of supertiles $\vec{\tilde{\alpha}} = (\tilde{\alpha}_i \mid 0 \leq i < k)$ is a *supertile assembly sequence* of $\mathcal{T}$ if there is an assembly sequence $\vec{S} = (S_i \mid 0 \leq i < k)$ of $\mathcal{T}$ such that, for all $1 \leq i < k$, $\text{succ}_{\vec{S}}(\tilde{\alpha}_{i-1}) = \tilde{\alpha}_i$, and $\vec{\tilde{\alpha}}$ is *nascent* if $\vec{S}$ is nascent.

The *result* of a supertile assembly sequence $\vec{\tilde{\alpha}}$ is the unique supertile $\text{res}(\vec{\tilde{\alpha}})$ such that there exist an assembly $\alpha \in \text{res}(\vec{\tilde{\alpha}})$ and, for each $0 \leq i < k$, assemblies $\alpha_i \in \tilde{\alpha}_i$ such that $\text{dom}\ \alpha = \bigcup_{0 \leq i < k} \text{dom}\ \alpha_i$ and, for each $0 \leq i < k$, $\alpha_i \sqsubseteq \alpha$. For all supertiles $\tilde{\alpha}, \tilde{\beta}$, we write $\tilde{\alpha} \to_{\mathcal{T}} \tilde{\beta}$ (or $\tilde{\alpha} \to \tilde{\beta}$ when $\mathcal{T}$ is clear from context) to denote that there is a supertile assembly sequence $\vec{\tilde{\alpha}} = (\tilde{\alpha}_i \mid 0 \leq i < k)$ such that $\tilde{\alpha}_0 = \tilde{\alpha}$ and $\text{res}(\vec{\tilde{\alpha}}) = \tilde{\beta}$. It can be shown using the techniques of [46] for seeded systems that for all two-handed tile assembly systems $\mathcal{T}$ supplying

[2] with the convention that $\infty = \infty + 1 = \infty - 1$

www.manaraa.com

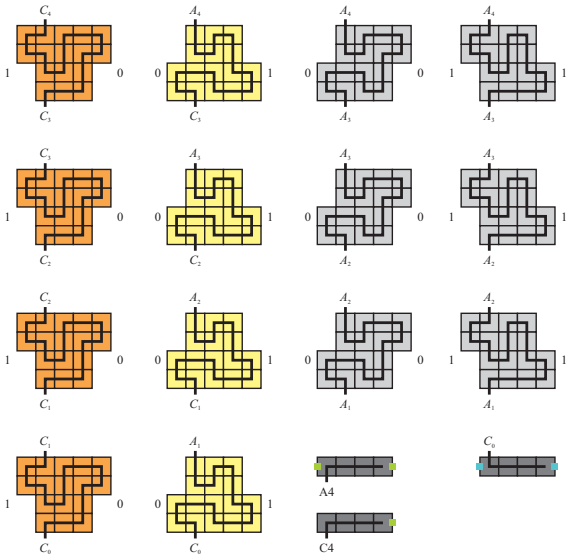


Figure 6.1: Tile set for two-handed assembly of a length n binary counter using $O(\log n)$ tile types.

an infinite number of each tile type, $\rightarrow_{\mathcal{T}}$ is a transitive, reflexive relation on supertiles of $\mathcal{T}$.

A supertile $\tilde{\alpha}$ is *producible*, and we write $\tilde{\alpha} \in \mathcal{A}[\mathcal{T}]$, if it is the result of a nascent supertile assembly sequence. A supertile $\tilde{\alpha}$ is *terminal* if, for all producible supertiles $\tilde{\beta}$, $C^{\tau}_{\tilde{\alpha},\tilde{\beta}} = \varnothing$.[3] Define $\mathcal{A}_{\square}[\mathcal{T}] \subseteq \mathcal{A}[\mathcal{T}]$ to be the set of terminal and producible supertiles of $\mathcal{T}$. $\mathcal{T}$ is *directed* (a.k.a., *deterministic*, *confluent*) if $|\mathcal{A}_{\square}[\mathcal{T}]| = 1$.

Let $X \subseteq \mathbb{Z}^2$ be a shape. We say X *strictly self-assembles* in $\mathcal{T}$ if, for all $\tilde{\alpha} \in \mathcal{A}_{\square}[\mathcal{T}]$, there exists $\alpha \in \tilde{\alpha}$ such that dom $\alpha = X$; i.e., $\mathcal{T}$ uniquely assembles into the shape X.

6.3 Two-Handed Assembly of a Counter from $O(\log n)$ Tile Types

In this section we describe the two-handed assembly of a (non-fault-tolerant) counter from $O(\log n)$ tile types, as a warmup to our full fault-tolerant square construction. While this technique does not achieve fault-tolerance, it introduces a novel new counter design technique that utilizes 1) geometry to enforce/restrict specific assemblies and 2) non-determinism of

[3]Note that a supertile $\tilde{\alpha}$ could be non-terminal in the sense that there is a producible supertile $\tilde{\beta}$ such that $C^{\tau}_{\tilde{\alpha},\tilde{\beta}} \neq \varnothing$, yet it may not be possible to produce $\tilde{\alpha}$ and $\tilde{\beta}$ simultaneously if some tile types are given finite initial counts, implying that $\tilde{\alpha}$ cannot be "grown" despite being non-terminal. If the count of each tile type in the initial state is ∞, then all producible supertiles are producible from any state, and the concept of terminal becomes synonymous with "not able to grow", since it would always be possible to use the abundant supply of tiles to assemble $\tilde{\beta}$ alongside $\tilde{\alpha}$ and then attach them.

www.manaraa.com

supertile attachment to explore the space of possible intermediate assemblies, despite the existence of only one unique terminal assembly. This technique forms the basis for the more involved fuzzy fault tolerant construction.

The tile set for the counter is depicted in Figure 6.1. In this figure, an example tile set for a 4 bit counter that counts from 0 to 15 is provided. Tile types that share unique, full strength $\tau = 2$ glues are connected by a black line that crosses over the bonded edge. Other glues in the system include strength $\tau = 2$ glues A_i and C_i for i from 0 to $\log n$ for a length n counter, and two strength $\tau = 1$ glues denoted by the green and blue squares. Conceptually, the

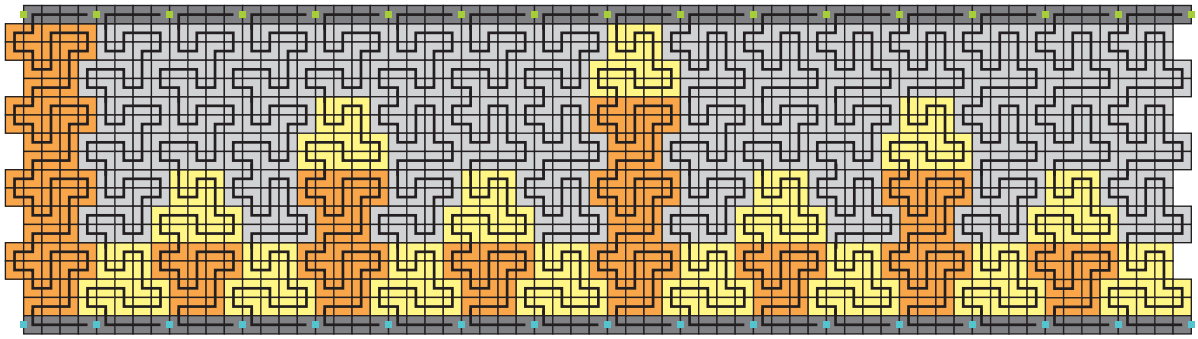


Figure 6.2: This is the fully assembled counter from the tile set given in Figure 6.1

.

tile set of Figure 6.1 consists of a number of blocks for each bit position of a binary counter. These blocks assemble into height $O(\log n)$ columns, where the representative block for each bit is determined non-deterministically. Further, the geometry of each block encodes a bit on both the left and right side of the block by a *dent* that appears at either the upper or lower half of the block. In the case of red blocks, or *rollover* blocks, the left side encodes the value 1, while the right encodes the value 0. For the yellow blocks, or *least significant 0* blocks, the left dent encodes the value 0 and the right encodes 1. For the grey blocks, or *copy* blocks, the left and right encode the same value, with one type of grey block for "1" and another for "0".

The glue types that connect blocks from one row to another ensure that any assembled column consists of red blocks from rows 1 to r (r at least 1 and at most $\log n$), followed by a yellow block in row $r + 1$ (if $r < \log n$), followed by grey blocks (either type) in rows $r + 2$ to row $\log n$ (if $r + 1 < \log n$). This pattern has the property that for any $(\log n)$-bit string

www.manaraa.com

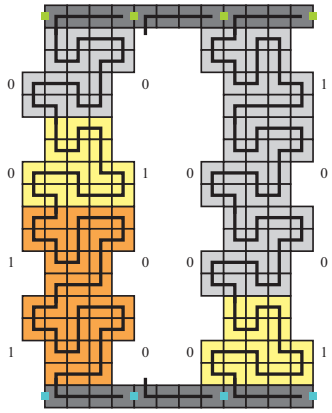


Figure 6.3: The basic temperature $\tau = 2$ counter in this section is not fuzzy fault tolerant. The above supertile is producible at temperature $\tau = 1$, stable at temperature $\tau = 2$, and cannot grow into the desired unique temperature $\tau = 2$ final assembly of Figure 6.2.

b, a column may assemble that encodes that string in the geometry of the dents on the left side of the column, and the right side of the column in turn encodes $b + 1$. Additionally, a fully assembled column can also attach the two four-tile chains of Figure 6.1 to both the top A glue and bottom C glue of the column. For any two assembled columns, the strength $\tau = 1$ green and blue glues combined give a strength $\tau = 2$ affinity for any two assembled columns to attach. However, due to the rigid teeth-like geometry of the columns, only sequential columns can get close enough to realize the affinity and assemble under the two-handed assembly model. Therefore, the unique assembly of the example tile set of Figure 6.1 is the counter shown in Figure 6.2.

In the example provided, we are specifically considering the special case of a counter that grows to a power of 2 length. More generally, it is possible to assemble only columns that encode values greater or equal to a given initial value, thereby allowing the assembly of a length-n counter for general n. However, we leave these details for the extended fault tolerant version of the construction.

The counter in this section is not fuzzy fault tolerant. In particular, the supertile in Figure 6.3 is producible at temperature $\tau = 1$ (but not $\tau = 2$ because the two-handed model requires that at most 2 supertiles, both of which are stable at $\tau = 2$, combine in any step), stable at temperature $\tau = 2$, but cannot grow into the correct unique $\tau = 2$ assembled counter

www.manaraa.com

of Figure 6.2.

6.4 Fuzzy Temperature Fault-Tolerance

In this section we introduce the fuzzy temperature model of fault-tolerance in self-assembly. The fuzzy temperature assembly model permits rampant temperature $\tau = 1$ growth of supertiles under the two-handed assembly model. We are then interested in what producible temperature $\tau = 1$ assemblies become stable at temperature $\tau = 2$. If even a single temperature $\tau = 1$ assembly becomes stable at temperature $\tau = 2$ and is inconsistent with what can be built in a purely temperature $\tau = 2$ assembly model, the system is deemed *error prone*. On the other hand, if all temperature $\tau = 1$ assemblies that are stable at temperature $\tau = 2$ have a valid temperature $\tau = 2$ path of growth to a supertile that is producible under a pure temperature $\tau = 2$ model, then the system is deemed *fuzzy temperature fault-tolerant*. Put another way, even with arbitrary erroneous strength 1 attachments, a fuzzy temperature fault-tolerant system guarantees that such errors cannot stabilize at temperature 2 unless the stabilized supertile can itself grow into a *correct* temperature $\tau = 2$ assembly, which means such an assembly is not really an error.

Formally, for a given initial tile set T, we define fuzzy temperature fault-tolerance in terms of the following four sets of supertiles: (1) The *dependably produced* (DP) supertiles are those that can be assembled at temperature $\tau = 2$ under the two-handed assembly model. Formally, DP is the set of all producible supertiles for the two-handed assembly system $(T, 2)$; (2) The *dependably terminal* (DT) supertiles are all supertiles in DP that cannot grow any further at temperature $\tau = 2$. Formally, DT is the set of terminal, producible supertiles for the two-handed assembly system $(T, 2)$; (3) The *plausibly produced* (PP) supertiles are those that can be assembled at temperature $\tau = 1$. Formally, PP is the set of all producible supertiles for the two-handed assembly system $(T, 1)$; and (4) The *plausibly stable* (PS) supertiles are all supertiles in PP that are stable at temperature $\tau = 2$.

Intuitively, DT denotes a final collection of supertiles that can be expected to be built given enough time for assembly in a temperature 2 system. On the other hand, due to the occasional

www.manaraa.com

assembly of supertiles with only strength 1 attachments, elements in PP will (plausibly) be assembled. Elements of PP that are not stable at temperature $\tau = 2$ intuitively will eventually break apart and are not of concern. However, these assemblies may grow to a point in which they become stable at temperature $\tau = 2$, in which case they will not break apart. Such assemblies constitute the set PS. The goal is to design a system such that for each element α of PS, every terminal β into which α can grow at temperature $\tau = 2$ is an element of DT (written $PS \Rightarrow DT$), and that DT is the set of desired shapes to be assembled. Put another way, we want to avoid the design of an error prone system in which stable assemblies that are inconsistent with the desired final assembly are built by erroneous $\tau = 1$ strength attachments.

More precisely, the fuzzy temperature fault-tolerance design problem is as follows:

fuzzy temperature fault-tolerance design problem: Given a target shape Υ, the goal is to design a tile set such that: (1) $PS \Rightarrow DT$ (fuzzy temperature fault-tolerance constraint); and (2) all supertiles in DT have shape Υ. (Desired goal shape is the unique output of the assembly.)

For the remainder of this paper, we attempt to solve the fuzzy temperature fault-tolerance problem for the benchmark example of an $n \times n$ square. As a metric, we are interested in minimizing the number of distinct tile types required to assemble a square while adhering to the fuzzy temperature fault-tolerance constraint. We show that a sleek $O(\log n)$ tile complexity is achievable, which is very close to the $O\left(\frac{\log n}{\log \log n}\right)$ that can be achieved with no fault-tolerance constraint.

6.5 Overview of Fault-Tolerant Square Construction

This section gives a high-level description of the main construction of this paper, a square that assembles under the fuzzy temperature fault tolerance model.

www.manaraa.com

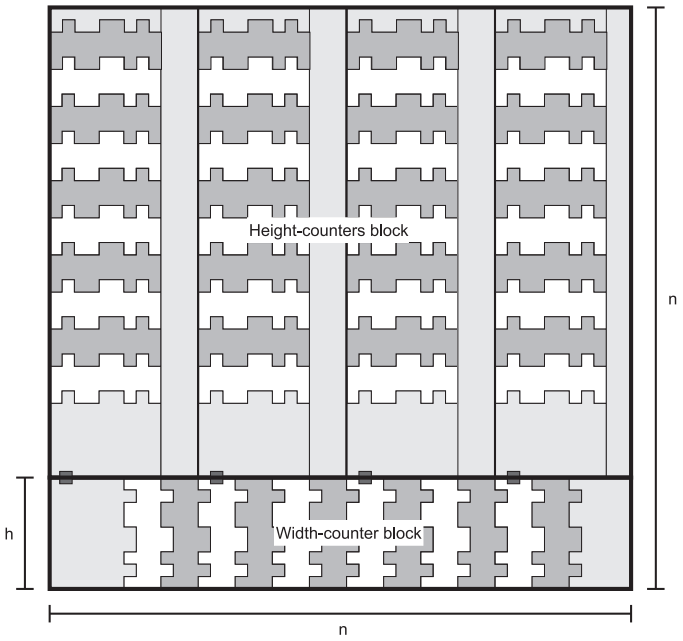


Figure 6.4: A simplified diagram of the components of a full $n \times n$ square. Components are not represented to scale.

6.5.1 Square

As is common in many self-assembly constructions for square-building, most of the work is in constructing counters that calculate the dimensions of the square. Figure 6.4 shows a high-level diagram of how to compose these counters. The horizontal counter and the vertical counters are constructed in conceptually the same way, with minor differences in the actual implementation. Most of the effort of our main construction is in encoding the number n into the tiles that grow a counter, so that it can control the length to which the counter grows, in a fault-tolerant way.

6.5.2 Counter

For simplicity we describe only the horizontal counter. The vertical counters are constructed similarly, with the exception that they are slightly simpler because of the need for the horizontal counter to correctly space out its bonds designed to connect the horizontal counter to the various vertical counters.

www.manaraa.com

Define $k \equiv \lfloor \log n \rfloor + 2$ be 1 plus the number of bits in n. As in Section 6.3, the counter consists of $\approx n$ columns (actually n divided by the width in tiles of a column, which is a constant, but for simplicity of discussion we will assume that there are n columns), each representing an integer between $2^k - n$ and $2^k - 1$. Note that we refer to columns as "counter-values." Each counter-value is connected to the next by two strength-1 *inter-counter-value glues*, and correct inter-counter-value binding is enforced using bumps and dents as in Section 6.3.

6.5.3 Counter-Value

As in Section 6.3, counter-values form randomly from $\approx \log n$ "bit gadgets", each of constant size, with each bit selected at random. Figure 6.6 shows the bit gadgets, and Figure 6.7 shows some of them attaching to form a few counter-values of a counter. Beyond the need for fuzzy temperature fault-tolerance, these bit gadgets must meet additional requirements. We first describe how to meet these requirements, and then describe how to achieve fault-tolerance.

6.5.3.1 Glue Design for Additional Requirements of Counter-Values

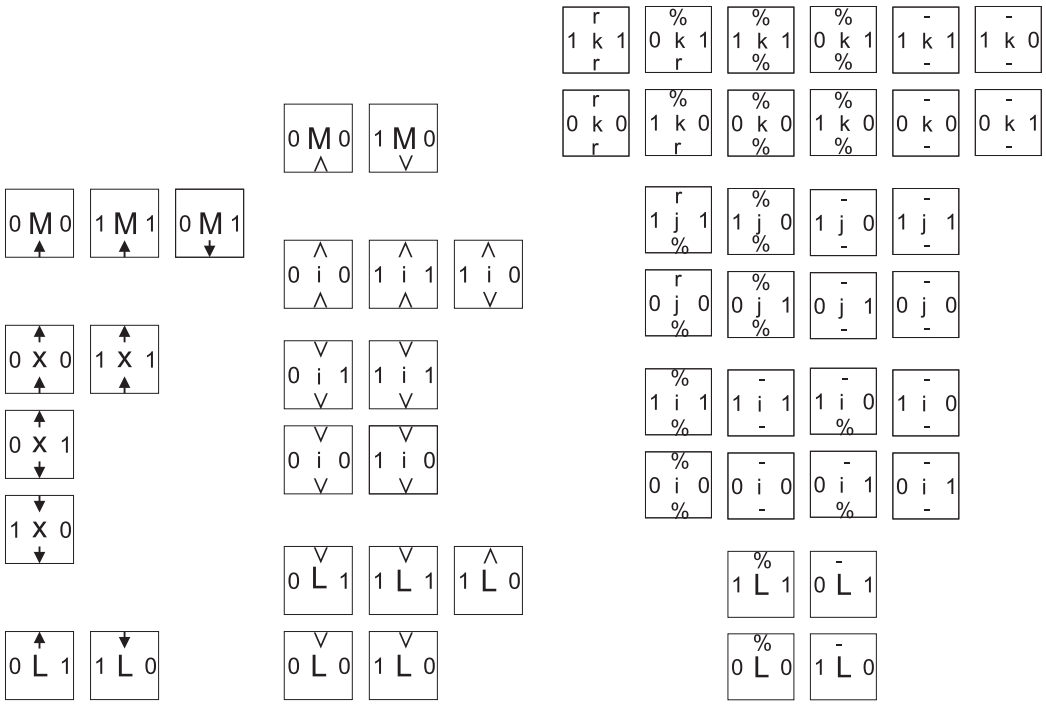
The logical requirements that counter-values must meet are:

1. The right side of a counter-value must represent $i + 1$ if the left side represents i. This was already needed in Section 6.3.

2. Each counter-value must be guaranteed to form an integer in the range $\left[2^k - n, 2^k - 1\right]$, so that the counter has exactly n counter-values.

3. Only a subset of appropriately spaced counter-values should have glues on the north to allow the vertical counters to bind, since the horizontal width of each vertical counters is $\Theta(\log n)$, whereas the horizontal width of each counter-value in the horizontal counter is $O(1)$. This is done by choosing a power of two 2^m (for m just large enough that $2^m >$ width of a vertical counter), and placing the glues to the north every 2^m counter-values.

www.manaraa.com



(a) Increment a binary number

(b) Enforces that the left number is greater than the right

(c) Detects if the binary number formed 1) is equal to a given number, or 2) matches a given number of the low-order bits of that number

Figure 6.5: Templates for tile sets that perform subsets of the functionality of the hairpin gadgets. Though not shown, each tile has strength-2 glues on the north and south, implemented in the actual tile set as a pair of single-strength bonds for fault-tolerance purposes. The east and west "bit values" in Figure 6.5a are represented in the actual tile set by the geometric shape of the 16×16 tile bit gadget that each individual tile in this figure represents, geometrically enforcing agreement on the bits of adjacent bit gadgets.

www.manaraa.com

The fault-tolerance is achieved entirely through the geometric design of the bit gadgets, and the choice of binding paths within them. The requirements (a), (b), and (c) are achieved through careful selection of the north-south glues that connect bit gadgets to each other. For the sake of meeting these three requirements, we can therefore logically view each bit gadget as a single tile, with double-strength glues on the north and south. The values of these glues will then be carried through to every actual tile that makes up a bit gadget, and combined with the glues that hard-code the relative position of each tile in the bit gadget, allow us to conceptually separate the problem of fault tolerance from that of meeting the three requirements discussed above. Finally, we can conceptually separate these three problems from each other, designing tiles to meet those requirements separately, and combine them in a cross-product construction. Figure 6.5 shows the three tile sets that meet the requirements (a), (b), and (c). (A brief discussion follows, but see Section 6.6.2 for an in-depth description.)

In each case, we take care to ensure that the requirement is met no matter in which order the tiles aggregate. Nonetheless, it is easiest to describe their operation as though the northmost tile is first present, and the counter-value assembles north-to-south; i.e., most significant bit to least significant.

Figure 6.5a shows the tiles that implement incrementing to ensure that the east bits represent $i+1$ if the west bits represent i. If the position of the least significant 0 in i is p, then all bits at positions above p are equal, all bits at positions below p are 1 for i and 0 for $i+1$, and at position p the bit is 0 for i and 1 for $i+1$. Therefore the tiles nondeterministically guess a position p at which to make this transition, and enforce that all tiles above p have equal bits and all tiles at or below p obey the stated requirement.

Figure 6.5b shows the tiles that implement range-checking to ensure that the number i that is constructed is greater than $m \equiv 2^k - n$. (Since precisely k bits are assembled, $i < 2^k$.) Imagine comparing i to m starting at the most significant bit. We must enforce that there is at least one bit difference, and that in the position of most significance where there is a difference that the bit from i is 1 and the bit from m is 0. As before, the tiles nondeterministically guess at which position the first disagreement will occur. Below the first disagreement, the bits of i

www.manaraa.com

are selected nondeterministically. We chose the value of k so that we know n's most significant bit is 0; this helps to ensure, if tiles grow from south to north and have not yet enforced $i > m$, then the most significant bit of i may be chosen equal to 1 to enforce this.

Figure 6.5c shows the tiles that ensure that two single-strength glues designed to be an anchor point for vertical counters are placed on the top of a counter-value in the horizontal counter if and only if the counter-value is at an appropriate position to space the vertical counters out evenly. This is accomplished by first determining the number, r, which will be represented by the rightmost counter-value for which this will be the case. Then, whenever the number i represented by a counter-value shares the same least significant m bits with r, the northern glues are present to anchor a vertical counter. Additionally, in the special case where all bits of i match those of r, a pair of northern glues unique to that position are present, to ensure that the special case, rightmost vertical counter with the necessary padding to fill out the width to exactly n, can attach.

6.5.3.2 Geometric Design for Fault-Tolerance

On the assumption that the three requirements in the previous section can be met for each counter-value that forms, we now describe how to use geometry and "synchronization primitives" involving careful placement of glues to ensure that even at temperature 1, unwanted structures cannot grow that will be stable at temperature 2. Recall that at temperature 2, the counter-values of the counter of Section 6.3 enforce that binding between adjacent counter-values cannot occur until both counter-values are fully assembled; this occurs because the path (consisting of all strength-2 glues) from one single-strength inter-counter-value glue to another goes through every bump of the counter-value. Hence, to have both glues present, the entire counter-value must also be present.

Our construction enforces that no structure producible even at temperature 1 can stably attach to the east of counter-value i unless it contains enough of the bumps of its westmost counter-value to enforce that binding requires that counter-value to represent $i + 1$. This is enforced by the following constraint: every path (including strength-1 glues) connecting the

www.manaraa.com

two inter-counter-value glues of counter-value i that intersects any counter-value $j > i$, also passes through every bump of the counter-value $i + 1$. Therefore, enough of the leftmost counter-value of this structure is guaranteed to be present to ensure that it can only bind to the right of counter-value i if its leftmost counter-value represents $i + 1$.

To enforce that a path from some part of counter-value i to some part of counter-value $i+2$ must traverse the entire height of counter-value $i + 1$, we must enforce that a path traverses southward through the bumps of counter-value $i+1$, and then traverses northward again before moving on to counter-value $i + 2$. But since the path cannot "short-circuit" there must be no glues between the southward and northward paths except at the bottom of the counter-value. The bumps and dents on the east side of the southward path must be faithfully represented on the east side of the northward path.

Even though the bits can grow in any order, it is easiest to imagine growing the bits of the southward path, then turning around and guessing those same bits while growing the northward path. Each bit along a single path is represented by what we will call a *hairpin gadget*; one southward and one northward hairpin gadget (though unconnected to each other) form a single bit gadget. To ensure that improper guesses do not result in junk assemblies that cannot grow any further, we use a similar motif to the "single-strength glues at opposite ends" used in Section 6.3, within the hairpin gadgets themselves. That is, hairpin gadgets can only bind stably to the north of other hairpin gadgets when fully formed, which prevents a hairpin gadget that does not match its complementary hairpin gadget from locking in. Figure 6.7 shows part of a counter formed from these gadgets. The white hairpin gadgets are "southward growing" (again, if we imagine tracing a path from counter-value i to counter-value $i + 1$, bearing in mind that the two-handed assembly can grow in other orders), and the gray hairpin gadgets are "northward growing".

Intuitively, the only connection between a white "left-half" of a counter-value and the gray "right-half" of that counter-value is through the southern row. Northward growth from this row is kept consistent by ensuring that no hairpin gadget can stabilize to the hairpin gadget beneath it until the purple double-bond is present. Since every path from this purple double-

www.manaraa.com

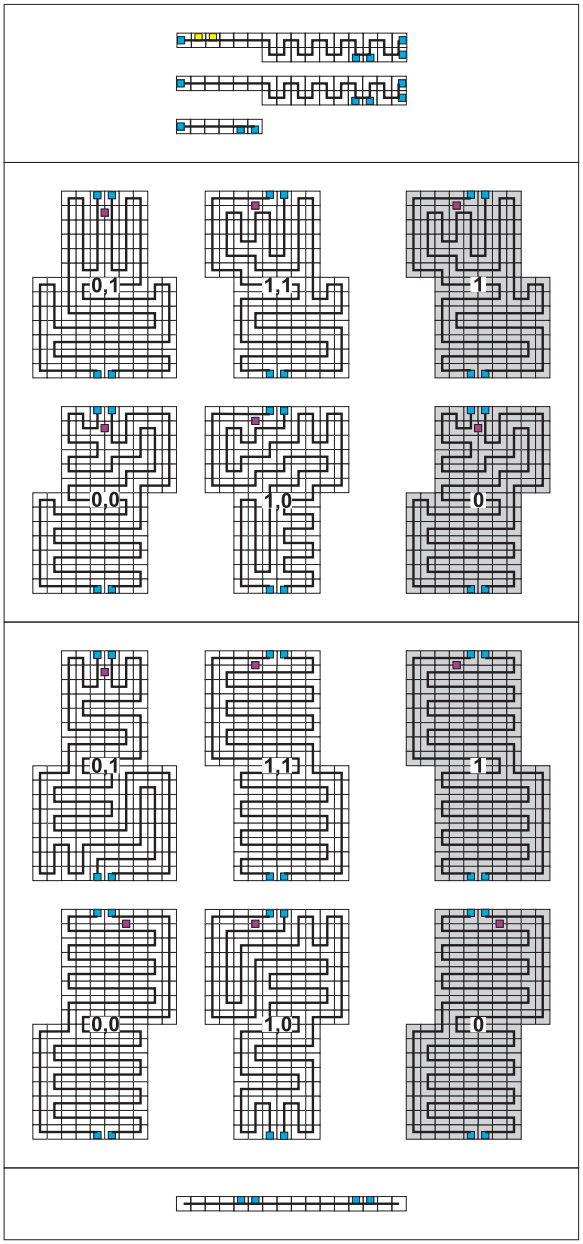


Figure 6.6: The gadgets that combine to form the counter-values of a counter. The top 6 gadgets that are labeled with bit values are of height 13 rather than 16 for the others, and are used only for the most significant bit in a counter value in order to compensate for the 3 rows of tiles necessary for the gadgets that attach to the top and bottom of the counter and hold the counter values together. Blue squares represent strength-1 bonds that bind hairpin gadgets to each other and the top/bottom gadgets. Purple squares represent the strength-2 bonds that connect the two paths (represented by dark black lines) formed by double strength bonds and forming the bump and dent patterns to represent bit values.

www.manaraa.com

bond to a blue single-bond on the south of the same hairpin gadget goes through the bumps of that gadget, the gadget cannot stabilize unless it is consistent with what has already grown to the left or right of it (and if nothing has already, then *it* determines what must be consistent with it).

Conversely, southward growth, which can lock a hairpin gadget to the hairpin gadget to its north *without* necessarily agreeing with the hairpin to its left or right, nevertheless cannot stabilize at temperature 2 without growing enough of those bumps to enforce agreement. This is because the bottom row must be present to connect a white counter-value half to its gray half, and both must be present to connect that counter-value to the previous (left) counter-value.

Finally, the bumps and dents at the eastmost and westmost edges leave some space that must be filled in if we wish to create a true square, and to create an exactly $n \times n$ square for any n there is additional 'padding' necessary. In order to obtain smooth edges for the counter, the first and last counter-value assemblies are created from hard-coded sets of tiles which have no bumps and dents on their left and right sides, respectively. Finally, additional hard-coded rows of the necessary length for the full width to be exactly n attach to the right side of the last counter value. (An example of such padding is shown in Figure 6.8.) In a similar way, the spacing between vertical counters is filled in.

6.6 Fault-Tolerant Assembly of a Square with $O(\log n)$ Tile Types

Figure 6.4 shows a high-level depiction of the main logical components, which create an $n \times n$ square. The square consists of two logical components: a bottom portion, which we will call the "width-counter block," and a top portion called the "height-counters block." The width-counter block is essentially a counter that calculates the width of the square. This counter is composed of a series of fixed width (16 tiles wide) columns, each representing a binary value x on its left side and the value $x + 1$ in its right side. The height-counter blocks are composed of a series of vertically oriented counters (similar to the horizontal counter in the width-counter block) that count to a lesser value so that their height (with the necessary padding) is n minus the height of the width-counter block. These counters are designed so

www.manaraa.com

that they attach to the north side of the width-counter block at specified locations.

Below are quantities that (collectively) define the dimensions and attachment locations of various sub-assemblies, the combinations of which form the completed square.

- n: the value of the height and width of the square to be assembled
- 16: the width and height of a "bit gadget," which is a sub-assembly representing a single bit in a value of a counter, this consists of the combination of a white hairpin gadget and its complementary grey hairpin gadget (see Figure 6.6). Note that the bit gadgets for the most significant bits are 16 tiles wide but only 13 tiles tall in order to compensate for the 2 extra rows of tiles on the top and 1 on the bottom which are used to connect the counter values.
- $c^{\rightarrow} = \lfloor n/16 \rfloor$, the number of counter values, or binary values to be counted by the width-counter, where a counter value is represented by the vertical combination of bit gadgets. $O(\log n)$ tile types are required to form these counter values.
- $p^{\rightarrow} = n - 16c^{\rightarrow}$, the extra width, or padding, added to the rightmost counter value of the width-counter to ensure that the width of the square is exactly n (the range is 0-15, thus requiring a constant sized set of tile types which form rows of length $p^{\rightarrow}$ that attach to the right edge of the rightmost counter value of the width-counter. See Figure 6.8 for an example.)
- $k^{\rightarrow} = 2^{\lceil \log c^{\rightarrow} \rceil} - c^{\rightarrow}$, the starting counter value for the width-counter, which will count from $k^{\rightarrow}$ to the next power of two, $2^{\lceil \log c^{\rightarrow} \rceil}$. The counter value $k^{\rightarrow}$ is actually formed by a hard-coded set of tiles (requiring $O(\log n)$ tile types), and the first counter value which actually assembles from bit gadgets is $k^{\rightarrow} + 1$.
- $k^{\rightarrow}_{Max} = 2^{\lceil \log c^{\rightarrow} \rceil} - 1$, the maximum counter value reached by the width-counter (consisting of all 1's in binary). Similar to the counter value for $k^{\rightarrow}$, this counter value is also represented by a hard-coded assembly requiring $O(\log n)$ tile types and which also has glues on the right side that allow the padding rows of length $p^{\rightarrow}$ to attach.

www.manaraa.com

- $b^{\rightarrow} = \lceil \log c^{\rightarrow} \rceil$, the number of bits in each width-counter value.
- $h = 16b^{\rightarrow}$ the height of the width-counter block, this accounts for the $b^{\rightarrow}$ bit gadgets composing each counter value
- $c^{\uparrow} = \lfloor \frac{n-h}{16} \rfloor$, the number of counter values to be counted by a height-counter
- $p^{\uparrow} = n - (h + 16c^{\rightarrow})$, the extra height, or padding, added to the topmost counter value of each height-counter to ensure that the width of the square is exactly n. The space and tile type requirements for this are similar to those for $p^{\rightarrow}$.
- $k^{\uparrow} = 2^{\lceil \log c^{\rightarrow} \rceil} - c^{\rightarrow}$, the starting counter value for each height-counter. The representation of this value in the assembly is analogous to that of $k^{\rightarrow}$.
- $k^{\uparrow}_{Max} = 2^{\lceil \log c^{\rightarrow} \rceil} - 1$, the maximum counter value reached by each height-counter (consisting of all 1's in binary). The representation of this value in the assembly is analogous to that of $k^{\rightarrow}_{Max}$.
- $b^{\uparrow} = \lceil \log c^{\rightarrow} \rceil$, the number of bits in each height-counter value
- $w^{\uparrow} = 16 \cdot 2^{\lceil \log b^{\uparrow} \rceil}$, the width of a padded-height-counter, which is a height-counter plus the necessary padding on its right side to fill in the gap between it and the height-counter to its immediate right (must be a power of two multiplied by 16 to allow the width-counter to provide regular binding sites on its north side for padded-height-counters to attach to)
- $pad = w^{\uparrow} - 16b^{\uparrow}$, the width of the padding on the right side of each height-counter (except for the rightmost, which is a special case). This width is $16 \cdot 2^{\lceil \log \lceil \log \lfloor \frac{n-h}{16} \rfloor \rceil \rceil} - 16b^{\uparrow}$, which is $O(\log n)$, and therefore the number of tile types required to make rows of that length are $O(\log n)$
- $\lfloor \frac{n}{w^{\uparrow}} \rfloor$, the number of padded-height-counters which attach to the north side of the width-counter to form the height-counters block

www.manaraa.com

- $pad^{\dashv} = n \mod w^{\uparrow}$, the width of the extra padding on the right side of the rightmost height-counter. This quantity is bounded by $2w^{\uparrow} - 16b^{\uparrow} - 1$, and therefore by $O(\log n)$, and thus the number of tile types required to form rows of this length are $O(\log n)$.
- $m = \log w^{\uparrow} = \lceil \log b^{\uparrow} \rceil$, the number of low-order bits to be matched in the width-counter values to determine the locations to initiate height-counters
- $r = k^{\rightarrow} + \left(w^{\uparrow} \frac{\left\lfloor \frac{n}{w^{\uparrow}} \right\rfloor - 1}{16} \right)$, the rightmost counter value of the width-counter which initiates a height-counter (this height-counter is a special case)

Note that no single component of the construction requires more than $O(\log n)$ tile types, whence the tile complexity of the entire construction is $O(\log n)$.

6.6.1 Hairpin Gadgets

The *hairpin gadgets* are the assemblies in the middle two boxes of Figure 6.6. Each hairpin gadget is composed of two single-tile-wide paths. These paths are denoted by the thick black lines that represent a series of tiles connected by double-strength bonds. Note that only tile edges through which one of these lines pass, or that contain a colored square, have a non-zero strength glue. The purple squares represent a double-strength bond that is the single point of connection between the two paths in each hairpin gadget. The blue squares, two on both the north and south edges of each hairpin gadget, represent single-strength bonds that enable hairpin gadgets to connect to each other in a vertical row.

The top box of hairpin gadgets in Figure 6.6 (in the second box from the top) are used to represent the most significant bits of the counter and are exactly 13 tiles tall. We construct copies of the other hairpin gadgets that are specific to each of the other bit positions and are 16 tiles tall. Bits are represented as "bumps" and "dents" on the east and west sides of the hairpin gadgets, so that vertical combinations of gadgets create patterns of teeth corresponding to binary numbers. The representation of each bit consists of both a bump and a dent (it is precisely this pattern that determines which bit is being represented in a particular location).

www.manaraa.com

We purposely design the two paths that snake through each hairpin gadget to extend the distance of at least one tile into each bump of that gadget.

The white hairpin gadgets form into a column that logically increments a binary counter by representing a binary number x on the west side of the column and $x+1$ on the east side. The grey hairpin gadgets simply present the same binary number on both sides and their utility will be discussed later.

It is clear that a column of white hairpin gadgets can bind to a column of grey hairpin gadgets via their bottom edge in such a way that the resulting structure is stable at temperature-2. In this case, since the purple squares joining the two paths of a given hairpin gadget are located near the north side of the gadgets, only a hairpin gadget which complementarily matches its neighbor can form and attach to the growing assembly. This ensures that no assembly sequence could result in a "junk" assembly in our construction under the rules of the fuzzy temperature model. Note that this is only a concern in the case where a column of hairpin gadgets is growing "up" from the bottom, because in the top-down growth case, they cannot be stably attached to the other hairpin gadgets (see the top box of Figure 6.6).

6.6.2 Logical Operation of Hairpin Gadgets

Logically, a hairpin gadget can be thought of as behaving as a scaled-up version (with scale-factor 16) of a single tile. The north and south glues act as strength-2 glues, but are combinations of the two strength-1 glues on the respective sides of the tiles labeled with blue squares on each of those edges. Due to the fault-tolerant design of the hairpin gadgets, the only way that an entire hairpin gadget can form as a stable assembly at temperature 2 is for both of the north glues to match each other, and for both of the south glues to match each other. This essentially mimicks two individual strength 2 glues (*i.e.,* the only way that any two sub-assemblies of a hairpin gadget could stably bind together is for both portions to "agree" on the type of hairpin gadget, since we specifically design all of the interior glues in such a way that the tiles bind exclusively to tiles of the exact same type of hairpin gadget). The strength 2 glues along each of the paths through the hairpin gadget are hard coded to form

www.manaraa.com

exactly those paths and provide no logical functionality other than connecting the two sides of the scaled, logical tile together.

The geometric design of the 16×16 hairpin gadget achieves the fault-tolerance of our main result, which is that no "junk" assembly can stabilize at temperature 2, even when temperature 1 growth is allowed. But when viewed as a single logical tile, each hairpin gadget can be thought of as encapsulating several sets of additional functions beyond fault-tolerance. Figure 6.5 depicts three separate templates for tile sets, and most hairpin gadgets combine the functionality of two of these. White hairpin gadgets perform the combined functionality of the tile sets in Figure 6.5a and Figure 6.5b, while the grey hairpin gadgets, which assemble into the width-counter block, perform the combined functionality of the tile sets in Figure 6.5b and Figure 6.5c.

6.6.2.1 Increment Tiles

Figure 6.5a depicts a template for a tile set that is used to form columns that represent pairs of fixed-length binary numbers of the form x (whose bits are represented as the leftmost labels) and $x + 1$ (whose bits are represented as the rightmost labels). There are copies of the tile types in the middle group, labeled with an x in the center, specific to each bit position other than the most and least significant positions (the tile types represented in the figure are considered a template for a tile set since copies of each tile type whose center label is "x" would have to be generated to be unique to each such bit position). All east and west edges have zero-strength glues, and north and south edges labeled with arrows have double-strength glues that are specific to their bit ordering, allowing tiles that represent bits in each position of significance to bind only to the correct neighbors. The numbers and letters are simply for labeling purposes. It is important to note that this tile set assembles individual columns of tiles, whose labels represent binary values x and $x + 1$, *independent* of the order in which the columns come together.

www.manaraa.com

6.6.2.2 Tiles to Enforce "$x > k$"

Figure 6.5b depicts a template for a tile set that assembles columns representing pairs of fixed-length binary numbers of the form $x > k$, where x is a variable binary string whose bits are represented by the labels on left, and k is a fixed value whose bits are represented by the labels on the right. This is merely a template for a tile set since, in order to generate the actual tile types, an input value k must be specified. First, a copy of each of the tile types whose middle label is i must be created specifically for each bit position of k other than the most or least significant bits. Then, all of the tile types whose right labels do not match the bit of k corresponding to the significance of their position are discarded. The resulting tile set consists of tiles that combine (in any order in the two-handed assembly model) to form columns of height equal to the number of bits in k, but only in patterns so that the value represented by the string x is greater than k. The range of values that x can take on is $k+1, k+2, \ldots, 2^{|k|}-1$ (where $|k|$ is the length of the binary representation of k, since k could - and will in our construction - have a 0 as its most significant bit).

6.6.2.3 Tiles that Determine Where to Initiate Height-Counters

Figure 6.5c depicts a template for a tile set that assembles columns representing pairs of fixed-length binary numbers x and r, where x is a variable binary string whose bits are represented by the labels on the left, and r is a fixed value whose bits are represented by the values on the right. Additionally, on the north side of the top tile of each column, the glue will specify whether:

1. $x = r$,
2. the low-order b bits of x match those of r, or
3. neither of the previous two conditions are true (with the glue for case 1 always being presented if x fully matches r, rather than the glue for the second case.

In order to generate this tile set, the input values r and b must be specified. Then, for each of the bit positions $1, 2, \ldots, b-1$, copies of the tile types with the label i are created with glues

www.manaraa.com

specific to these bit positions. Next, a copy of each tile type with the label j is created so that the glues position them in the b^{th}-most significant position, and copies are made of the tile types with the label k for each of the remaining positions of greater significance. Finally, tile types whose right labels do not match the bit of r corresponding to the significance of their position are discarded. The resulting tile set consists of tiles that can assemble in any order in the two-handed assembly model to form columns of height equal to the number of bits in r. These columns also represent an arbitrary binary string x, with the north most glue of the column being "r" if $x = r$, "%" if the low-order b bits of x match those of r, and "-" otherwise.

6.6.2.4 Combining Tile Set Functionality Into the Hairpin Gadgets

As mentioned previously, most of the hairpin gadgets implement a combination of the functionality of two of the tile sets shown in Figure 6.5. The white hairpin gadgets perform the combined functionality of assembling into a representation of some binary string x whose value is greater than a given k ($k^{\rightarrow}$ for hairpin gadgets that assemble the width-counter and $k^{\uparrow}$ for those that assemble the height-counters) and less than or equal to the maximum value ($k^{\rightarrow}_{Max}$ for the width-counter and $k^{\uparrow}_{Max}$ for the height-counters), while representing the value of x on the left and $x+1$ on the right for the width-counter (and bottom and top for the height-counters). The grey hairpin gadgets that assemble into the height-counters perform only the functionality of forming values of x that are within the same ranges as those of their white gadget counterparts. However, the grey hairpin gadgets that assemble into counter values for the width-counter also combine the functionality of presenting glues on their northernmost edges which denote whether $x = r$, whether only the low-order m bits of x match those bits of r, or whether neither of these conditions hold. For counter values satisfying one of the first two aforementioned cases, a special top gadget (one of which is shown in the top box of Figure 6.6 with the two yellow squares on its north side) forms the north piece of that counter value. This is what allows the height-counters to connect in the correct positions.

In order to combine the functionality of two tile sets, tile types from each set are matched up according to the bit positions that they represent. Then, we perform a simple "cross product"

www.manaraa.com

by taking each pair of tile types representing the same bit position as well as the same value for x (*i.e.,* they have matching leftmost labels) and making a single, new tile type whose glues and labels contain all of the information from the glues and labels of the two original tile types (with only one copy of the label for x).

Finally, for each tile type t in such a generated set, in order to represent t as a hairpin gadget, a final transformation is necessary. For this example we will discuss how to convert a tile type t that (1) represents a bit position of x that is neither the least nor most significant and (2) that combines the "increment x" and "enforce $x > k$" functionality with values of 0 and 0 for the bits of x and $x + 1$, respectively. Note that transformations for the other classes of tile types is similar and therefore we omit a formal discussion. The form for the hairpin gadget to be constructed is that of the white hairpin gadget in the third box from the top in Figure 6.6 with the label "$0, 0$."

First, a tile type is created for the position at the southern end of the left path, containing a blue square. Its south glue is strength-2 and contains both arrows from the south glue of t (one of each coming from the tile sets to increment and compare), along with an "L" (since it will be part of the left path through the hairpin gadget), as well as a number representing the significance of the bit. The east and west sides have null glues; the glue for the north side is strength-2, and contains both of the necessary arrows plus the information that both bits are 0, the number representing the significance of the bit, and an 'L1' since it is the first tile on the left path. Next, we add a tile type for each of the interior positions in the left path through that hairpin gadget, with glues that match the north glue of that first tile type, but with each position incrementing the value with "L" so that each tile type is specific to its position in the path and can only attach to its neighbors along that path (also accounting for the direction of the path through each tile type and putting the glues on the appropriate edges). Next, a tile type for the north end of the path is created, analogous to that for the south tile. Then, in a similar matter, tile types for the right path are created (changing the "L" to an "R"). Finally, for the tile types that represent the positions on the two paths where the purple square is located, a strength-2 glue is placed that is unique to exactly those two tile types.

www.manaraa.com

In this manner, all of the tile types required for the hairpin gadgets of the width-counter and height-counters (whose tile types require a 90 degree counter-clockwise rotation from the corresponding tile types of the width-counter) can be generated for a tile set that will assemble into an $n \times n$ square. Notice that the information about the bits of x contained in the labels of each t are now converted to patterns of binary teeth that are used to enforce the correct order of assembly of counter sub-assemblies via geometric constraints.

6.6.3 Formation of a Bit Gadget

A *bit gadget* is a logical assembly consisting of a white hairpin gadget and its complementary grey hairpin gadget located on its east side so that all 16 tiles along the adjacent edges of each gadget are in contact with each other. Note that there are only zero-strength glues along this edges so the two hairpin gadgets do not bind to each other. Only hairpin gadgets that encode the same bit along their shared edges can form a bit gadget due to the geometric constraints imposed by the bumps and dents. For the sake of simplicity, hairpin gadgets are designed so that bit gadgets are effectively 16×16 squares (counting bumps on only one side).

6.6.4 Formation of a Counter-Value

There are two types of counters in our construction, the width-counter and a set of height-counters. Both types of counters are fixed-width counters. The width-counter counts from the value $k^{\rightarrow}$ to $k^{\rightarrow}_{Max}$, with each value represented by a binary string of $b^{\rightarrow}$ bits. The height-counters count from the value $k^{\uparrow}$ to $k^{\uparrow}_{Max}$, with each value represented by a binary string of $b^{\uparrow}$ bits. To speak generically of either type of counter, we will use b to denote the number of bits in the values represented by a particular counter. A *counter-value* is an assembly composed of exactly b bit gadgets, along with the top and bottom gadgets that are shown in the top and bottom boxes of Figure 6.6 (a counter-value is essentially a column). The design of each of the components that make up a counter-value are such that no matter the order in which they assemble, a full counter-value assembly can form only if a valid bit string within the range specified for the counter forms correctly.

www.manaraa.com

6.6.5 Formation of the counters

The width-counter is composed of $c^{\rightarrow} - 2$ counter-value sub-assemblies, along with a hard-coded assembly on the left side, which acts as a counter-value representing $k^{\rightarrow}$ via a smooth left side (containing no bumps or dents), and a hard-coded assembly on the right side representing $k^{\rightarrow}_{Max}$ via a smooth right side and rows of length $p^{\rightarrow}$ padding attached to the right. The height-counters are similar but rotated 90 degrees counterclockwise, with $c^{\rightarrow} - 2$ counter-value sub-assemblies, capped by values $k^{\uparrow}$ and $k^{\uparrow}_{Max}$ and northern padding of height $p^{\uparrow}$. Additionally, the height-counters have rows of length *pad* attached to their right sides (except for the rightmost such counter, whose padding rows are of length $pad^{\dashv}$). Figure 6.7 shows an example sub-assembly of the width-counter.

Only a fully and correctly formed counter-value (*i.e.,* one within the correct range and with the correct shape) can stably bind to another counter-value with the correct bit pattern due to the nature of the two types of top gadgets along with the geometry imposed by the bumps and dents representing bits. In this way, we ensure that counter values can connect to each other only in the correct order and also without "skipping" any columns of the counter. Therefore, the counters all grow to exactly the correct lengths and, moreover, no "junk" assemblies are allowed to stabilize at temperature-2.

6.6.6 Formation of the square

Although the components could actually come together in a number of possible orderings, it is valid to consider just one of them for simplicity of discussion due to the fault tolerant design of the tile set. Therefore, assuming that the width-counter and each of the height-counters correctly and completely form first, then the east side padding for the width-counter, consisting of rows of length $p^{\rightarrow}$, attach and the north side padding for each of the height-counters, consisting of columns of length $p^{\uparrow}$, attach. (Figure 6.8 shows an example of padding rows attaching.) Next, the east side padding for each of the height-counters, consisting of rows of length *pad* (or $pad^{\dashv}$ for the special case height-counter that attaches as the rightmost height-counter) attach to each height-counter. Finally, the height-counters attach to the northern edge

www.manaraa.com

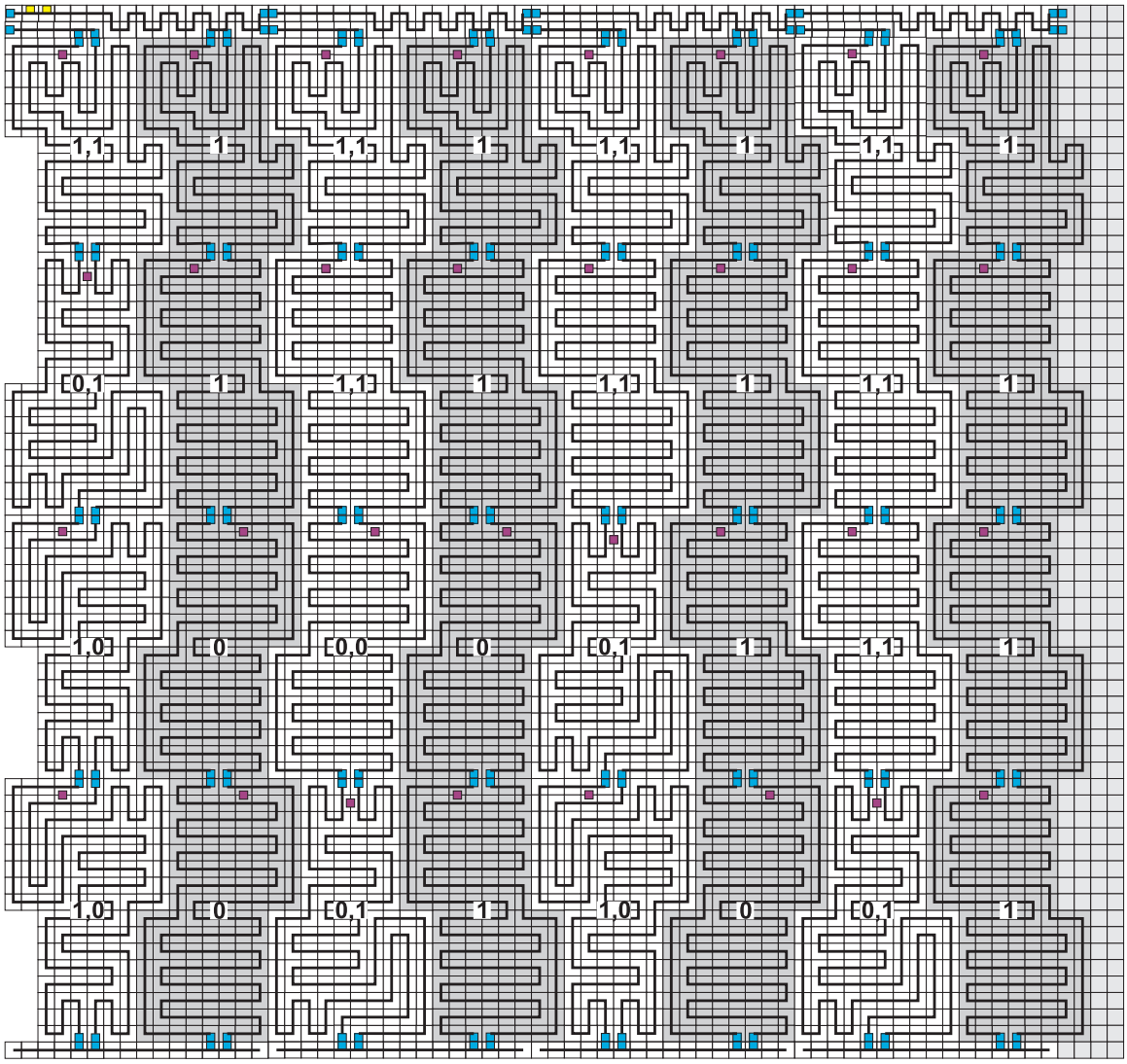


Figure 6.7: An example of the rightmost 4 columns of the width-counter, counting from 1100 to 1111.

www.manaraa.com

of the width-counter at the correct locations. The resulting assembly is exactly an $n \times n$ square.

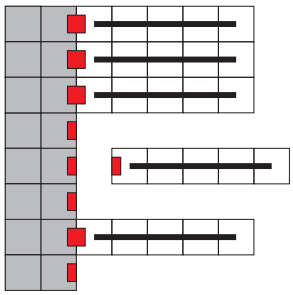


Figure 6.8: An example of padding rows. The grey portion represents the right side of an assembly to which padding must attach. The padding in this figure is of width 5 and depicted by the rows of white tiles, which are formed by 5 tile types that bind in exactly such a row that binds only on its left side to the correct sub-assembly. In this figure, 4 padding rows have attached and a fifth is nearly in position to do so.

6.7 Conclusion

Adleman, Cheng, Goel, and Huang [2] show that for each n there is a (seeded,non-fault-tolerant) tile assembly system that uniquely assembles an $n \times n$ square using $O(\log n / \log \log n)$ unique tile types, a bound that was shown asymptotically tight by Rothemund and Winfree [47]. Since our construction uses $\Theta(\log n)$ tile types, an obvious open question is whether there is a fuzzy temperature fault-tolerant tile assembly system that uses the asymptotically optimal $O(\log n / \log \log n)$ to uniquely assemble an $n \times n$ square.

Our construction is "floppy": many adjacent tiles in the final square are not connected by glues. One would expect that more strongly connected squares are more physically resilient, and they may also enforce the geometric blocking utilized heavily in our construction, so this floppiness may be a disadvantage. Given the goal of preventing all erroneous temperature-1 growth from stabilizing, it seems unlikely that a *full square* – a square in which every neighboring pair of tiles interact with positive strength – could be constructed using a fuzzy temperature fault-tolerant system. But it is conceivable that more elaborate use of synchronization could allow extra "support substructures" to be used to make our construction "more fully connected", while preserving the fuzzy temperature fault-tolerance.

www.manaraa.com

6.7.1 Is the Fuzzy Temperature Model Universal?

Since we have shown that $n \times n$ squares uniquely self-assemble in the fuzzy temperature model, a natural question to ask is the following: is the fuzzy temperature model Turing universal? In other words, is it possible to design a fuzzy temperature tile assembly system that uniquely simulates an arbitrary Turing machine on some input? At the time of this writing, it is not clear if accomplishing such an objective is possible. For instance, using a simulation technique similar to our fuzzy temperature counting scheme where the columns would represent configurations of a TM might not work. This is because, in the main construction presented in this chapter, our counter was able to assemble by counting "forward" or even "in reverse." If one were to simulate a TM in this fashion, then it would seem that junk assemblies resulting from "reverse simulation steps" would be able to stabilize at temperature 2 and therefore the construction would not uniquely produce a structure representing the simulation of some arbitrary TM. However, some other more exotic simulation technique might be able to show that the fuzzy temperature model is Turing universal. Nevertheless, we formally state this problem as an open question.

Question 27. Is the fuzzy temperature model Turing universal?

www.manaraa.com

BIBLIOGRAPHY

[1] Leonard Adleman, *Towards a mathematical theory of self-assembly*, Tech. report, University of Southern California, 2000.

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

[3] Gagan Aggarwal, Michael H. Goldwasser, Ming-Yang Kao, and Robert T. Schweller, *Complexities for generalized models of self-assembly*, Proceedings of ACM-SIAM Symposium on Discrete Algorithms, 2004.

[4] J. Albert and K. Čulik II, *A simple universal cellular automaton and its one-way and totalistic version*, Complex Systems **1** (1987), no. 1, 1–16.

[5] Jonathan Bachrach and Jacob Beal, *Building spatial computers*, Tech. report, MIT CSAIL, 2007.

[6] 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 **106** (2009), no. 15, 6054–6059.

[7] Jacob Beal and Gerald Sussman, *Biologically-inspired robust spatial programming*, Tech. report, MIT, 2005.

www.manaraa.com

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

[9] Ho-Lin Chen, Qi Cheng, Ashish Goel, Ming-Deh Huang, and Pablo Moisset de Espanes, *Invadable self-assembly: Combining robustness with efficiency*, SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms), 2004.

[10] Ho-Lin Chen and Ashish Goel, *Error free self-assembly with error prone tiles*, Proceedings of the 10th International Meeting on DNA Based Computers, 2004.

[11] Ho-Lin Chen, Ashish Goel, and Chris Luhrs, *Dimension augmentation and combinatorial criteria for efficient error-resistant DNA self-assembly*, Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008 (Shang-Hua Teng, ed.), SIAM, 2008, pp. 409–418.

[12] ———, *Dimension augmentation and combinatorial criteria for efficient error-resistant dna self-assembly*, SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms (Philadelphia, PA, USA), Society for Industrial and Applied Mathematics, 2008, pp. 409–418.

[13] Ho-Lin Chen, Rebecca Schulman, Ashish Goel, and Erik Winfree, *Reducing facet nucleation during algorithmic self-assembly*, Nano Letters **7** (2007), no. 9, 2913–2919.

[14] Qi Cheng, Gagan Aggarwal, Michael H. Goldwasser, Ming-Yang Kao, Robert T. Schweller, and Pablo Moisset de Espanés, *Complexities for generalized models of self-assembly*, SIAM Journal on Computing **34** (2005), 1493–1515.

[15] Qi Cheng, Ashish Goel, and Pablo Moisset de Espanés, *Optimal self-assembly of counters at temperature two*, Proceedings of the First Conference on Foundations of Nanoscience: Self-assembled Architectures and Devices, 2004.

www.manaraa.com

[16] 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** (2008), no. 3, 347–370.

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

[18] David Doty, Jack H. Lutz, Matthew J. Patitz, Scott M. Summers, and Damien Woods, *Intrinsic universality in self-assembly*, Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science, 2009, pp. 275–286.

[19] ———, *Random number selection in self-assembly*, Proceedings of The Eighth International Conference on Unconventional Computation (Porta Delgada (Azores), Portugal, September 7-11, 2009), 2009.

[20] David Doty, Matthew J. Patitz, and Scott M. Summers, *Limitations of self-assembly at temperature 1*, Proceedings of The Fifteenth International Meeting on DNA Computing and Molecular Programming (Fayetteville, Arkansas, USA, June 8-11, 2009), 2009, pp. 283–294.

[21] B. Durand and Zs. Róka, *The game of life: universality revisited*, Tech. Report 98-01, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, January 1998.

[22] Kenichi Fujibayashi, David Yu Zhang, Erik Winfree, and Satoshi Murata, *Error suppression mechanisms for dna tile self-assembly and their simulation*, Natural Computing: an international journal **8** (2009), no. 3, 589–612.

[23] Kenichi Fujibayashi, David Yu Zhang, Erik Winfree, and Satoshi Murata, *Error suppression mechanisms for dna tile self-assembly and their simulation*, Natural Computing: an international journal **8** (2009), no. 3, 589–612.

[24] P. Gács, *Reliable computation with cellular automata*, Journal of Computer and System Sciences **32** (1986), no. 1, 15–78.

www.manaraa.com

[25] J. Hartmanis and R. E. Stearns, *On the computational complexity of algorithms*, Transactions of the American Mathematical Society **117** (1965), 285–306.

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

[27] ———, *Randomized self-assembly for approximate shapes*, International Colloqium on Automata, Languages, and Programming (ICALP), Lecture Notes in Computer Science, vol. 5125, Springer, 2008, pp. 370–384.

[28] Grégory Lafitte and Michael Weiss, *Universal tilings*, STACS (Wolfgang Thomas and Pascal Weil, eds.), Lecture Notes in Computer Science, vol. 4393, Springer, 2007, pp. 367–380.

[29] ———, *Simulations between tilings*, Tech. report, University of Athens, 2008.

[30] ———, *An almost totally universal tile set*, TAMC (Jianer Chen and S. Barry Cooper, eds.), Lecture Notes in Computer Science, vol. 5532, Springer, 2009, pp. 271–280.

[31] James I. Lathrop, Jack H. Lutz, Matthew J. Patitz, and Scott M. Summers, *Computability and complexity in self-assembly*, Theory of Computing Systems, to appear.

[32] James I. Lathrop, Jack H. Lutz, and Scott M. Summers, *Strict self-assembly of discrete Sierpinski triangles*, Proceedings of The Third Conference on Computability in Europe (Siena, Italy, June 18-23, 2007), 2007.

[33] Paul Vitányi Ming Li, *An introduction to kolmogorov complexity and its applications*, Springer, 1997.

[34] Furong Liu, Ruojie Sha, and Nadrian C. Seeman, *Modifying the surface features of two-dimensional DNA crystals.*, Journal of the American Chemical Society **121** (1999), no. 5, 917–922.

www.manaraa.com

[35] Urmi Majumder, Thomas H LaBean, and John H Reif, *Activatable tiles for compact error-resilient directional assembly*, 13th International Meeting on DNA Computing (DNA 13), Memphis, Tennessee, June 4-8, 2007., 2007.

[36] 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** (2000), no. 6803, 493–6.

[37] 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** (1999), no. 23, 5437–5443.

[38] Maurice Margenstern, *Cellular automata in hyperbolic spaces, vol. 1: Theory*, Old City Publishing, Philadelphia, 2007.

[39] Nicolas Ollinger, *The intrinsic universality problem of one-dimensional cellular automata*, 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS) (H. Alt and M. Habib, eds.), LNCS, vol. 2607, Springer, 2003, pp. 632–641.

[40] ———, *Intrinsically universal cellular automata*, Proceedings International Workshop on The Complexity of Simple Programs, Cork, Ireland, 6-7th December 2008 (T. Neary, D. Woods, A.K. Seda, and N. Murphy, eds.), EPTCS, vol. 1, 2009, arXiv:0906.3213v1 [cs.CC], pp. 199–204.

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

[42] John Reif, Sudheer Sahu, and Peng Yin, *Compact error-resilient computational DNA tiling assemblies*, DNA: International Workshop on DNA-Based Computers, LNCS, 2004.

[43] John H. Reif, *Local parallel biomolecular computing*, DNA Based Computers III, volume 48 of DIMACS, American Mathematical Society, 1999, pp. 217–254.

www.manaraa.com

[44] ———, *Molecular assembly and computation: From theory to experimental demonstrations*, Proceedings of the Twenty-Ninth International Colloquium on Automata, Languages and Programming, 2002, pp. 1–21.

[45] P. W. K. Rothemund, *Design of dna origami*, ICCAD '05: Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design (Washington, DC, USA), IEEE Computer Society, 2005, pp. 471–478.

[46] Paul W. K. Rothemund, *Theory and experiments in algorithmic self-assembly*, Ph.D. thesis, University of Southern California, December 2001.

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

[48] Paul W.K. Rothemund, Nick Papadakis, and Erik Winfree, *Algorithmic self-assembly of DNA Sierpinski triangles*, PLoS Biology **2** (2004), no. 12, 2041–2053.

[49] Rebecca Schulman and Erik Winfree, *Programmable control of nucleation for algorithmic self-assembly*, DNA: International Workshop on DNA-Based Computers, LNCS, 2004.

[50] ———, *Synthesis of crystals with a programmable kinetic barrier to nucleation*, Proceedings of the National Academy of Sciences **104** (2007), no. 39, 15236–15241.

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

[52] David Soloveichik, Matthew Cook, and Erik Winfree, *Combining self-healing and proofreading in self-assembly*, Natural Computing **7** (2008), no. 2, 203–218.

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

[54] ———, *Complexity of self-assembled shapes*, SIAM Journal on Computing **36** (2007), no. 6, 1544–1569.

www.manaraa.com

[55] Thomas LaBean Urmi Majumder, Sudheer Sahu and John H. Reif, *Design and simulation of self-repairing DNA lattices*, DNA Computing: DNA12, Lecture Notes in Computer Science, vol. 4287, Springer-Verlag, 2006.

[56] Hao Wang, *Proving theorems by pattern recognition – II*, The Bell System Technical Journal **XL** (1961), no. 1, 1–41.

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

[58] Erik Winfree, *Algorithmic self-assembly of DNA*, Ph.D. thesis, California Institute of Technology, June 1998.

[59] ———, *Simulations of computing by self-assembly*, Tech. Report CaltechCSTR:1998.22, California Institute of Technology, 1998.

[60] ———, *Self-healing tile sets*, Nanotechnology: Science and Computation (Junghuei Chen, Natasa Jonoska, and Grzegorz Rozenberg, eds.), Natural Computing Series, Springer, 2006, pp. 55–78.

[61] Erik Winfree and Renat Bekbolatov, *Proofreading tile sets: Error correction for algorithmic self-assembly.*, DNA (Junghuei Chen and John H. Reif, eds.), Lecture Notes in Computer Science, vol. 2943, Springer, 2003, pp. 126–144.

[62] Erik Winfree, Furong Liu, Lisa A. Wenzler, and Nadrian C. Seeman, *Design and self-assembly of two-dimensional DNA crystals.*, Nature **394** (1998), no. 6693, 539–44.

www.manaraa.com