5-2019

Powers and Behaviors of Directed Self-assembly

Trent Allen Rogers
University of Arkansas, Fayetteville

Powers and Behaviors of Directed Self-assembly

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computer Science

by

Trent Rogers
University of Arkansas
Bachelor of Science in Mathematics, 2014

May 2019
University of Arkansas

This dissertation is approved for recommendation to the Graduate Council.

Matthew Patitz, Ph.D.
Dissertation Director

Wing-Ning Li , Ph.D.
Committee Member

Jin-Woo Kim, Ph.D.
Committee Member

Xintao Wu, Ph.D.
Committee Member

Abstract

In nature there are a variety of self-assembling systems occurring at varying scales which give rise to incredibly complex behaviors. Theoretical models of self-assembly allow us to gain insight into the fundamental nature of self-assembly independent of the specific physical implementation. In Winfree's abstract tile assembly model (aTAM), the atomic components are unit square "tiles" which have "glues" on their four sides. Beginning from a seed assembly, these tiles attach one at a time during the assembly process in an asynchronous and nondeterministic manner.

We can gain valuable insights into the nature of self-assembly by comparing different models of self-assembly which use fundamentally different mechanisms for local interactions. A powerful notion which allows us to compare models of self-assembly is simulation. The first result of this thesis examines the role of non-determinism in simulation. It shows that the universal simulation of directed aTAM systems requires undirectedness. A tile assembly model is said to be directed if it always assembles the same final assembly.

We distinguish between two types of aTAM systems: cooperative systems and non-cooperative systems. In cooperative aTAM systems, we are able to enforce that in order for a tile to attach to an assembly, the glues of a tile must match two or more glues of neighboring tiles. On the other hand, in non-cooperative aTAM systems, tiles are able to attach to an assembly provided that one of the tile's glues match an exposed glue on the assembly. It is well known that the cooperative aTAM is computationally universal, and it is conjectured that the non-cooperative aTAM is not computationally universal. For our second result, we show that if we allow tiles to be polygons with six or more sides, then the class of non-cooperative systems is capable of universal computation. On the other hand, we show that the class of systems consisting of polygons with six or less sides is not capable of computing using any of the currently known methods.

Acknowledgements

I would like to give a special thanks to my adviser, Matthew Patitz. Matt was a great mentor both in research and outside of research. He taught me how to work hard and ingrained a work ethic in me which will aid me throughout my career. Matt introduced me to a community of great people who I had the pleasure of collaborating with. His support allowed me to travel around the world to collaborate with a wonderful group of people.

In addition, I would like to thank Jacob Hendricks whose presence greatly improved my experience in graduate school. Jacob's one of the most intelligent people that I have ever met. He had a profound impact on the way I approach problems and evaluate proofs. His work ethic inspired me to work harder, and we spent many nights in the lab together during marathon working sessions. It was inspiring to see his combination of brilliance and work ethic pay off when he received an offer for a tenure track position. Above all, he was a great friend and really made graduate school a lot more enjoyable.

Finally, thanks to the entire self-assembly community. Dave Doty's encyclopedic knowledge and elegant rigor inspired me to expand my knowledge and improved my proof writing abilities. Shinosuke Seki was a great mentor and always a welcoming host during my visits to Tokyo. Damien Woods is one of the most kind, genuine people that I have met and is always willing to listen to my ideas on new research directions.

Dedication

I would like to thank my parents for their support throughout my educational career.

Contents

1 Introduction **1**
- 1.1 Self-assembly . . . 1
- 1.2 Simulation in self-assembly . . . 2
- 1.3 Directed self-assembly . . . 3
- 1.4 Dissertation overview . . . 3
  - 1.4.1 The directed aTAM is not intrinsically universal . . . 4
  - 1.4.2 The non-cooperative polygonal tile assembly model is computationally universal . . . 4

2 Preliminaries **6**
- 2.1 Informal description of the abstract tile assembly model . . . 6
- 2.2 Formal description of the abstract tile assembly model . . . 7

3 A class of systems that is not intrinsically universal **10**
- 3.1 Introduction . . . 10
- 3.2 Preliminaries . . . 12
  - 3.2.1 Simulation . . . 12
  - 3.2.2 Intrinsic universality . . . 14
- 3.3 The directed aTAM is not intrinsically universal . . . 15
- 3.4 Overview of the directed aTAM system $\mathcal{T}$ . . . 15
  - 3.4.1 Overview of modules of $\mathcal{T}$ . . . 17
  - 3.4.2 Directedness of $\mathcal{T}$ . . . 21
- 3.5 Overview of impossibility of simulation . . . 22
- 3.6 Details of the directed system $\mathcal{T}$ . . . 24
  - 3.6.1 Languages and Turing machines used . . . 24
  - 3.6.2 `planter` . . . 27
  - 3.6.3 `left` . . . 29
  - 3.6.4 `right` . . . 31
  - 3.6.5 `top` . . . 31
  - 3.6.6 `arm` . . . 32
  - 3.6.7 `bitAlley` . . . 32
  - 3.6.8 Summary of computations . . . 33
  - 3.6.9 The system $\mathcal{T}$ is directed . . . 33
- 3.7 Details of impossibility of simulation . . . 34
  - 3.7.1 Empty subiterations cannot be uniquely marked in advance . . . 35
  - 3.7.2 Turing machines simulating tile assembly systems . . . 41
  - 3.7.3 A contradiction . . . 44
- 3.8 Technical lemmas . . . 49
  - 3.8.1 Miscellaneous definitions . . . 49
  - 3.8.2 Path-crossing subconfigurations . . . 50
  - 3.8.3 Necessity of probes . . . 52
  - 3.8.4 Narrowing down the outputs of a set of Turing machines . . . 56
  - 3.8.5 Zig-zag assembly systems . . . 60
  - 3.8.6 Space complexity of zig-zag systems is invariant under simulation . . . 60

4 A computationally universal, non-cooperative model **79**
4.1 Introduction . . . 79
4.2 Preliminaries . . . 80
4.3 Geometric bit-reading, grids, and Turing machine simulation . . . 83
4.3.1 Main results . . . 83
4.3.2 Bit-reading gadgets overview . . . 85
4.3.3 Formal definition of bit-reading gadget . . . 87
4.3.4 Grid assemblies . . . 88
4.3.5 Turing machine simulation . . . 89
4.4 Regular polygonal tile analysis with complex roots . . . 93
4.4.1 Complex roots of unity example using heptagonal tiles . . . 95
4.5 Overview of polygonal grid construction . . . 97
4.6 Polygonal grid construction . . . 98
4.6.1 Grid notation . . . 111
4.6.2 *Normalized* bit-reading gadgets . . . 111
4.7 Polygons which "can not compute" at temperature 1 . . . 111
4.7.1 Equilateral triangles, squares, and regular hexagons . . . 112
4.7.2 Regular pentagons . . . 116
4.8 Bit-reading gadgets . . . 120
4.8.1 Single shape systems with regular polygonal tiles . . . 120
4.8.2 2-shaped systems with regular polygonal tiles . . . 128
4.8.3 Single shaped systems with equilateral polygonal tiles . . . 129
4.8.4 A single shaped system with triangular tiles . . . 131
4.9 Building *normalized* bit-reading gadgets . . . 131
4.9.1 Constructing on grid bit-writer configurations . . . 134
4.9.2 Connecting the bit-writer subconfigurations . . . 139
4.9.3 Normalizing bit-writers . . . 141
4.9.4 Shifting on grid after the read . . . 144
4.9.5 Proof of correctness . . . 144
4.10 Technical appendix . . . 145
4.10.1 Systems with tiles shaped like a single regular polygon . . . 145
4.10.2 2-shaped systems with regular polygonal tiles . . . 166
4.10.3 Single shaped systems with equilateral polygonal tiles . . . 169

5 Conclusion and future work **170**
5.1 Intrinsic universality of systems with varying levels of nondeterminism . . . 170
5.2 Simulation in the polygonal TAM . . . 171

References **173**

List of published papers

- Jacob Hendricks, Matthew J. Patitz, and Trent A. Rogers. Universal simulation of directed systems in the abstract tile assembly model requires undirectedness. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2016), New Brunswick, New Jersey, USA* October 9-11, 2016, pages 800-809. Chapter 3. Published.

- Oscar Gilbert, Jacob Hendricks, Matthew J. Patitz, and Trent A. Rogers. Computing in continuous space with self-assembling polygonal tiles. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016), Arlington, VA, USA* January 10-12, 2016, pages 937-956, 2016. Chapter 4. Published.

Chapter 1

Introduction

1.1 Self-assembly

Self-assembly is the process by which components autonomously interact and coalesce without the intervention of a centralized, controlling force. Self-assembly can be found in a variety of settings from the nano-scale interactions which allow for the formation of biological organisms to the gravitational interactions between stars which lead to the formation of galaxies. Studying self-assembly can provide insights into a wide range of topics including the origin of life [38] and bottom-up manufacturing [24, 34, 1].

In order to study the theoretical properties of self-assembling systems, we use an axiomatic system which abstracts the specifics of the underlying implementation. This allows us to draw conclusions about self-assembling systems in a generalized manner which is independent of the exact mechanics of the implementation. In this dissertation, we focus on a set of models which are colloquially known as tile assembly models. In these models, the atomic components are modeled as "tiles". Informally, a tile is a polygon where each side of the polygon has a string associated with it which we call a "glue". The glues on a tile and a system parameter called the "temperature" determine if a tile is able to attach to an existing assembly. The behavior and structures created by these systems are then able to be controlled by "programming" the glues on the tiles.

These models are based on Erik Winfree's abstract tile assembly model (aTAM) [39]. The aTAM restricts the tiles to be squares, and models the assembly process as beginning from a "seed" assembly and proceeding non deterministically and asynchronously with single tile attachments. One of the distinguishing features of the aTAM is that a number of aTAM systems have been experimentally implemented in the laboratory [39, 40] including systems which leverage algorithmic self-assembly [37, 12]. These systems have been implemented using carefully designed DNA "tiles" which use single-stranded DNA to mimic the glues of the tiles. The ability to physically implement algorithmic self-assembly allows

us to build complex structures using relatively few unique tile types. This makes building complex structures in the laboratory easier and reduces the cost of building these structures. For example, the non-algorithmic approach to building an $n \times n$ square requires n^2 tile types while the algorithmic approach only requires $O(\frac{\log n}{\log \log n})$ [36].

1.2 Simulation in self-assembly

As experimentalists continue to advance technology, they are presented with an increasing number of options and trade-offs when designing self-assembling systems. Is it possible to make design choices which are optimal in terms of the structures which are able to feasibly assemble? In order to discuss optimal design choices, we need a way to compare models. The notion of simulation provides us the necessary framework to compare different system design choices which give rise to different models.

Intuitively, we say that system $\mathcal{S}$ simulates a system $\mathcal{T}$ provided that, under some block replacement scheme, $\mathcal{S}$ behaves the same as $\mathcal{T}$, and consequently produces the same assembly as $\mathcal{T}$ modulo the scale factor of the blocks used. If every system in model B can be simulated by some system in model A, then it is natural to intuitively consider model A as being at least as "powerful" as model B. Furthermore, if there are systems in A which cannot be simulated by any system in model B, then it is natural to consider model A as being more powerful than model B. This stems from model A being able to do anything that model B can do (up to scale factor) and more.

A more stringent requirement where one tile set (when properly seeded) must be able to simulate all systems in a model is known as intrinsic universality. Intrinsic universality has been studied in numerous models of computation [30, 11, 25] including the aTAM in which it was shown the aTAM is intrinsically universal[9]. In terms of tile assembly models, we say that a tile set S is intrinsically universal (IU) for a class of systems $\mathcal{C}$ provided that for every system $\mathcal{T} \in \mathcal{C}$, there exists some input assembly α composed of tiles from S such that when S is seeded with α, it simulates $\mathcal{T}$.

In addition to assisting in physical design choices, the notion of simulation has been shown to be mathematically interesting. In [41], Woods provides a survey of universal simulation in tile assembly models. Woods shows a rich, complex relationship between the different tile assembly models as well as demonstrates the robustness of the definition of simulation in tile assembly models. Models which at first seem incomparable are able to be measured relative to one another when compared using the notion of simulation. One of the major insights of [41] is to show the complex hierarchy of tile assembly models that arises when using universal simulation as a measure of comparison.

1.3 Directed self-assembly

Understanding the role of nondeterminism in a variety of models of computation has been a central focus of several works [5, 22]. One of the most well known open problems in theoretical computer science (namely, does $P = NP$?) centers around understanding the role nondeterminism plays in Turing machines. Unsurprisingly, the role nondeterminism plays in self-assembly has attracted a lot of attention. Of particular interest is understanding how the power of undirected systems relate to directed systems. Informally, a tile assembly system is said to be directed if the assembly process always results in the same final assembly.

1.4 Dissertation overview

This dissertation focuses on two types of simulation: 1) simulation between tile assembly models and 2) simulation of Turing machines by tile assembly systems. The first set of results show an impossibility result relating to the simulation between tile assembly systems while the second result shows a positive result relating to the simulation of Turing machines by a non-cooperative tile assembly model. In non-cooperative tile assembly systems, tiles are able to attach to an assembly provided that just one of the tile's glues matches an exposed glue on the assembly.

1.4.1 The directed aTAM is not intrinsically universal

The construction presented in [9] fundamentally requires nondeterminism in the form of undirectedness even to simulate directed systems. In [3], it was shown that there are certain shapes which can be assembled by undirected systems that cannot be assembled by any directed system. Could it also be the case that the class of undirected systems are also "more powerful" than the class of directed systems when discussing simulation? Clearly, directed systems cannot simulate the class of undirected systems since simulating the behavior of undirected systems requires undirectedness by definition. Thus, the interesting scenario to consider is whether the class of directed systems requires undirectedness for any universal simulator.

In our first main result we show that nondeterminism plays a fundamental role in the universal simulation of the class of directed aTAM systems. That is, we show that there is no universal simulator for the class of directed systems which is itself always directed. In proving this result, we leverage results from structural computational complexity to provide a suite of new tools for proving impossibility results in tile assembly models. In particular, we prove a foundational result which shows that no valid simulating system can use asymptotically more computational resources than the system that it is simulating. We use this in conjunction with strict versions of the time and space hierarchies to sufficiently confuse any directed universal simulator and force it to resort to nondeterminism resulting in undirectedness.

1.4.2 The non-cooperative polygonal tile assembly model is computationally universal

In [39], it was shown that the aTAM with cooperation is computationally universal. In this context, cooperation refers to the constraint that at least two glues on a tile must match the glues of its neighboring tiles in order to bind to an assembly. This allows the system to simulate a Turing machine by only allowing a tile to bind to the assembly if its "input

sides" match both the current state of the machine and the input symbol which the head is currently reading. Physically realizing systems which rely on cooperation presents a number of challenges and requires carefully balancing a number of experimental parameters. Unfortunately, it is conjectured that the non-cooperative aTAM is incapable of algorithmic growth [10].

In [4], the first non-cooperative tile assembly model capable of universal computation was demonstrated. Cook et al. introduced a three-dimensional extension of the aTAM and showed that for every Turing machine M, there exists a non-cooperative 3D aTAM system capable of simulating M. A 2D variant of the aTAM was introduced in [21] called the dupled aTAM which lifted the restriction that all tiles must be squares and allowed for 2×1 rectangular tiles. Hendricks et al. then showed the non-cooperative dupled aTAM is capable of universal computation. The dupled aTAM was further relaxed in [13] to allow any mixture of polyominoes as tile shapes. Let P be a polyomino of size three or greater. In the main result, Hendricks et al. showed that for every Turing machine M, there exists a system consisting of tiles of shape P capable of simulating M without using cooperation.

In our second set of results, we define the polygonal tile assembly model (polygonal TAM). Informally, this model is an extension of the aTAM where the only restriction on the shape of the tiles is that they are polygons. Let P be a regular polygon with seven or more sides. We begin by showing that for every Turing machine M, there exists a directed, non-cooperative polygonal TAM system consisting of tiles of shape P capable of simulating M. We then show that if P is a regular polygon with six or less sides, the class of non-cooperative systems with tiles of shape P cannot compute using currently known techniques. This result relies on fundamentally different techniques than those used in [4, 21, 13] since the assembly process occurs in continuous space as opposed to the integer lattice. The lack of a regular underlying grid presents significant challenges.

Chapter 2

Preliminaries

In this chapter, we provide an intuitive description of the abstract Tile Assembly Model (aTAM) as well as a formal definition of the model. For more information about the development of the aTAM see [36, 39, 27].

2.1 Informal description of the abstract tile assembly model

This section gives a brief informal sketch of the abstract Tile Assembly Model (aTAM). See Section 2.2 for a formal definition of the aTAM.

A *tile type* is a unit square with four sides, each consisting of a *glue label*, often represented as a finite string, and a nonnegative integer *strength*. A glue g that appears on multiple tiles (or sides) always has the same strength s_g. There are a finite set T of tile types, but an infinite number of copies of each tile type, with each copy being referred to as a *tile*. An *assembly* is a positioning of tiles on the integer lattice $\mathbb{Z}^2$, described formally as a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$. Let $\mathcal{A}^T$ denote the set of all assemblies of tiles from T, and let $\mathcal{A}^T_{<\infty}$ denote the set of finite assemblies of tiles from T. We write $\alpha \sqsubseteq \beta$ to denote that α is a *subassembly* of β, which means that dom $\alpha \subseteq$ dom β and $\alpha(p) = \beta(p)$ for all points $p \in$ dom α. Two adjacent tiles in an assembly *interact*, or are *attached*, if the glue labels on their abutting sides are equal and have positive strength. Each assembly induces a *binding graph*, a grid graph whose vertices are tiles, with an edge between two tiles if they interact. The assembly is τ*-stable* if every cut of its binding graph has strength at least τ, where the strength of a cut is the sum of all of the individual glue strengths in the cut.

A *tile assembly system* (TAS) is a triple $\mathcal{T} = (T, \sigma, \tau)$, where T is a finite set of tile types, $\sigma : \mathbb{Z}^2 \dashrightarrow T$ is a finite, τ-stable *seed assembly*, and τ is a positive integer called the *temperature*. An assembly α is *producible* if either $\alpha = \sigma$ or if β is a producible assembly and α can be obtained from β by the stable binding of a single tile. In this case we write $\beta \rightarrow^{\mathcal{T}}_1 \alpha$ (to mean α is producible from β by the attachment of one tile), and we

write $\beta \rightarrow^{\mathcal{T}} \alpha$ if $\beta \rightarrow_1^{\mathcal{T}*} \alpha$ (to mean α is producible from β by the attachment of zero or more tiles). When $\mathcal{T}$ is clear from context, we may write $\rightarrow_1$ and $\rightarrow$ instead. We let $\mathcal{A}[\mathcal{T}]$ denote the set of producible assemblies of $\mathcal{T}$. An assembly is *terminal* if no tile can be τ-stably attached to it. We let $\mathcal{A}_\square[\mathcal{T}] \subseteq \mathcal{A}[\mathcal{T}]$ denote the set of producible, terminal assemblies of $\mathcal{T}$. A TAS $\mathcal{T}$ is *directed* if $|\mathcal{A}_\square[\mathcal{T}]| = 1$. Hence, although a directed system may be nondeterministic in terms of the order of tile placements, it is deterministic in the sense that exactly one terminal assembly is producible (this is analogous to the notion of *confluence* in rewriting systems).

2.2 Formal description of the abstract tile assembly model

In this section we provide a set of definitions and conventions that are used throughout this thesis.

We work in the 2-dimensional discrete space $\mathbb{Z}^2$. Define the set

$$U_2 = \{(0,1),(1,0),(0,-1),(-1,0)\}$$

to be the set of all *unit vectors* in $\mathbb{Z}^2$. We also sometimes refer to these vectors by their cardinal directions N, E, S, W, respectively. All *graphs* in this thesis are undirected. A *grid graph* is a graph $G = (V, E)$ in which $V \subseteq \mathbb{Z}^2$ and every edge $\{\vec{a}, \vec{b}\} \in E$ has the property that $\vec{a} - \vec{b} \in U_2$.

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

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.

Fix a finite set T of tile types. A *T-assembly*, sometimes denoted simply as an *assembly* when T is clear from the context, 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.

We now give a brief formal definition of the aTAM. See [39, 36, 35, 27] for other developments of the model. Our notation is that of [27], which also contains a more complete definition.

Given a set T of tile types, an *assembly* is a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$. An assembly is *$\tau$-stable* if it cannot be broken up into smaller assemblies without breaking bonds of total strength at least τ, for some $\tau \in \mathbb{N}$.

Self-assembly begins with a *seed assembly* σ and proceeds asynchronously and nondeterministically, with tiles adsorbing one at a time to the existing assembly in any manner that preserves τ-stability at all times. A *tile assembly system* (*TAS*) is an ordered triple $\mathcal{T} = (T, \sigma, \tau)$, where T is a finite set of tile types, σ is a seed assembly with finite domain, and $\tau \in \mathbb{N}$. A *generalized tile assembly system* (*GTAS*) is defined similarly, but without the finiteness requirements. We write $\mathcal{A}[\mathcal{T}]$ for the set of all assemblies that can arise (in finitely many steps or in the limit) from $\mathcal{T}$. An assembly $\alpha \in \mathcal{A}[\mathcal{T}]$ is *terminal*, and we write $\alpha \in \mathcal{A}_\square[\mathcal{T}]$, if no tile can be τ-stably added to it. It is clear that $\mathcal{A}_\square[\mathcal{T}] \subseteq \mathcal{A}[\mathcal{T}]$.

An assembly sequence in a TAS $\mathcal{T}$ is a (finite or infinite) sequence $\vec{\alpha} = (\alpha_0, \alpha_1, \ldots)$ of assemblies in which each α_{i+1} is obtained from α_i by the addition of a single tile. The *result* res$(\vec{\alpha})$ of such an assembly sequence is its unique limiting assembly. (This is the last assembly in the sequence if the sequence is finite.) The set $\mathcal{A}[\mathcal{T}]$ is partially ordered by the relation $\longrightarrow$ defined by

$$\alpha \longrightarrow \alpha' \quad \text{iff} \quad \text{there is an assembly sequence } \vec{\alpha} = (\alpha_0, \alpha_1, \ldots)$$
$$\text{such that } \alpha_0 = \alpha \text{ and } \alpha' = \text{res}(\vec{\alpha}).$$

If $\vec{\alpha} = (\alpha_0, \alpha_1, \ldots)$ is an assembly sequence in $\mathcal{T}$ and $\vec{m} \in \mathbb{Z}^2$, then the *$\vec{\alpha}$-index* of $\vec{m}$ is $i_{\vec{\alpha}}(\vec{m}) = \min\{i \in \mathbb{N} | \vec{m} \in \text{dom } \alpha_i\}$. 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 $\vec{m} \in \bigcup_{0 \leq i \leq l} \text{dom } \alpha_i$, define the set of its input sides $\text{IN}^{\vec{\alpha}}(\vec{m}) = \{\vec{u} \in U_2 | \text{str}_{\alpha_{i_\alpha}(\vec{m})}(\vec{u}) > 0\}$.

We say that $\mathcal{T}$ is *directed* (a.k.a. *deterministic*, *confluent*, *produces a unique assembly*) if the relation $\longrightarrow$ is directed, i.e., if for all $\alpha, \alpha' \in \mathcal{A}[\mathcal{T}]$, there exists $\alpha'' \in \mathcal{A}[\mathcal{T}]$ such that $\alpha \longrightarrow \alpha''$ and $\alpha' \longrightarrow \alpha''$. It is easy to show that $\mathcal{T}$ is directed if and only if there is a unique terminal assembly $\alpha \in \mathcal{A}[\mathcal{T}]$ such that $\sigma \longrightarrow \alpha$.

A set $X \subseteq \mathbb{Z}^2$ *weakly self-assembles* if there exists a TAS $\mathcal{T} = (T, \sigma, \tau)$ and a set $B \subseteq T$ such that $\alpha^{-1}(B) = X$ holds for every terminal assembly $\alpha \in \mathcal{A}_\square[\mathcal{T}]$. Essentially, weak self-assembly can be thought of as the creation (or "painting") of a pattern of tiles from B (usually taken to be a unique "color") on a possibly larger "canvas" of un-colored tiles.

A set X *strictly self-assembles* if there is a TAS $\mathcal{T}$ for which every assembly $\alpha \in \mathcal{A}_\square[\mathcal{T}]$ satisfies dom $\alpha = X$. Essentially, strict self-assembly means that tiles are only placed in positions defined by the shape. Note that if X strictly self-assembles, then X weakly self-assembles. (Let all tiles be in B.)

Chapter 3

A class of systems that is not intrinsically universal

3.1 Introduction

The aTAM has been shown to be intrinsically universal (IU) [9], meaning that there exists a single tile set, U, such that given any arbitrary aTAM system $\mathcal{T}$, U can be given an initial configuration which will cause it to faithfully simulate the full dynamics of $\mathcal{T}$ modulo a constant scale factor (dependent on $\mathcal{T}$). Since the result of [9], several other results related to IU have been used to examine and classify the relative powers of a variety of models of self-assembly and classes of systems within them [7, 20, 14, 19, 18, 29, 16, 13, 6], thus developing a complexity hierarchy which can be used to categorize models and systems within them.

In this paper, we investigate the problem of characterizing the role of nondeterminism within the aTAM, which has previously been explored in a variety of different aspects [2, 8, 23]. At its core, the aTAM is an asynchronous and nondeterministic model in which tile attachments to a growing assembly, while constrained by the requirement that sufficient matching glues must bind, are random with respect to the sequence of locations and sometimes the particular types of tiles which bind. The amount of nondeterminism of different aTAM systems can vary wildly, with some systems having uncountably infinite sets of producible, or even terminal (i.e. those which cannot grow any further), assemblies and/or sequences of assembly, to those having exactly one producible assembly and even some with just one possible assembly sequence. This leads to questions about whether or not, and possibly how much, nondeterminism is required to give the aTAM its full power. In this paper, we focus on this question from the perspective of the "universal aTAM simulator" of [9], which by design has several so-called "points of competition", where different assembly sequences of the simulator, as it simulates a system $\mathcal{T}$, race to grow paths to those points, with the first path to arrive causing a tile type specific to that path to be placed. The fact that there are multiple assembly sequences, each growing a

different path first, causes nondeterminism in the types of tiles placed in these locations. The use of such locations is so fundamental to that universal simulator's design, allowing it to continue growth of portions of the assembly without having to rely on future paths which may or may not ever arrive, that even when it is simulating directed aTAM systems, which are those that have exactly one terminal assembly and only one possible tile type in any location regardless of the assembly sequence, the simulator itself must be undirected. It has remained unknown whether or not such nondeterminism is fundamentally required by a universal simulator, and in Theorem 1 we prove that it is. That is, we prove that the class containing all directed aTAM systems is not IU, meaning that there exists no tile set U such that, given an arbitrary directed aTAM system, U can be configured to create an aTAM system which simulates it while itself being directed. Stated another way, it means that any universal simulator for the aTAM must be more nondeterministic than some of the systems which it simulates.

While our main result presents key insights into the properties required of aTAM and other tile-based simulators, and shows how nondeterminism with respect to the selection of assembly sequences can force nondeterminism with respect to assemblies produced by any universal simulator, other key contributions of this paper include the development of several new system design techniques and tools useful in proving properties about the computational resources available to be harnessed by embedded algorithms, which themselves provide additional insights into the computations possible using static combinations of matter filling non-reusable space. More specifically, we make use of computational complexity results which combine extremely tight worst-case and best-case space complexity bounds for decidable languages [33], as well as novel techniques for controlling the "input bandwidth" and geometries of carefully designed subassemblies which perform complex computations that are effectively hidden from each other. These designs are likely to be useful in further tile-based self-assembly results, especially impossibility results. Furthermore, we develop several important and potentially very useful tools which can be used to

characterize properties of tile assembly systems which are simulating others, e.g. Lemma 9 which proves that the space complexity of computations which can be performed by a system simulating a type of system known as a zig-zag system is asymptotically no greater than that of the computations which can be performed by the original system, despite the scale factor allowed the simulator.

Section 3.2 provides a set of preliminary definitions used throughout the paper, and the following section a formal statement of our main result. Next are two sections dedicated to a high-level overview of the proof, with sections including the full technical details following.

3.2 Preliminaries

In this section we define what it means for one tile assembly system to simulate another, and the notion of intrinsic universality.

3.2.1 Simulation

To state our main results, we must formally define what it means for one TAS to "simulate" another. Our definitions come from [29]. Intuitively, simulation of a system $\mathcal{T}$ by a system $\mathcal{S}$ requires that there is some scale factor $m \in \mathbb{Z}^+$ such that $m \times m$ squares of tiles in $\mathcal{S}$ represent individual tiles in $\mathcal{T}$, and there is a "representation function" capable of inspecting assemblies in $\mathcal{S}$ and mapping them to assemblies in $\mathcal{T}$.

From this point on, let T be a tile set, and let $m \in \mathbb{Z}^+$. An *m-block supertile* over T is a partial function $\alpha : \mathbb{Z}^2_m \dashrightarrow T$, where $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$. Let B^T_m be the set of all m-block supertiles over T. The m-block with no domain is said to be *empty*. For a general assembly $\alpha : \mathbb{Z}^2 \dashrightarrow T$ and $(x_0, x_1) \in \mathbb{Z}^2$, define $\alpha^m_{x_0,x_1}$ to be the m-block supertile defined by $\alpha^m_{x_0,x_1}(i_0, i_1) = \alpha(mx_0 + i_0, mx_1 + i_1)$ for $0 \leq i_0, i_1 < m$. For some tile set S, a partial function $R : B^S_m \dashrightarrow T$ is said to be a *valid m-block supertile representation* from S to T if for any $\alpha, \beta \in B^S_m$ such that $\alpha \sqsubseteq \beta$ and $\alpha \in \text{dom } R$, then $R(\alpha) = R(\beta)$.

For a given valid m-block supertile representation function R from tile set S to tile set T, define the *assembly representation function*[1] $R^* : \mathcal{A}^S \to \mathcal{A}^T$ such that $R^*(\alpha') = \alpha$ if and only if $\alpha(x_0, x_1) = R\left(\alpha'^m_{x_0,x_1}\right)$ for all $(x_0, x_1) \in \mathbb{Z}^2$. For an assembly $\alpha' \in \mathcal{A}^S$ such that $R(\alpha') = \alpha$, α' is said to map *cleanly* to $\alpha \in \mathcal{A}^T$ under R^* if for all non empty blocks $\alpha'^m_{x_0,x_1}$, $(x_0, x_1) + (u_0, u_1) \in \text{dom } \alpha$ for some $u_0, u_1 \in U_2$ such that $u_0^2 + u_1^2 \leq 1$, or if α' has at most one non-empty m-block $\alpha^m_{0,0}$.

In other words, α' may have tiles on supertile blocks representing empty space in α, but only if that position is adjacent to a tile in α. We call such growth "around the edges" of α' *fuzz* and thus restrict it to be adjacent to only valid supertiles, but not diagonally adjacent (i.e. we do not permit *diagonal fuzz*).

In the following definitions, let $\mathcal{T} = (T, \sigma_T, \tau_T)$ be a TAS, let $\mathcal{S} = (S, \sigma_S, \tau_S)$ be a TAS, and let R be an m-block representation function $R : B^S_m \to T$.

Definition. We say that $\mathcal{S}$ and $\mathcal{T}$ have *equivalent productions* (under R), and we write $\mathcal{S} \Leftrightarrow \mathcal{T}$ if the following conditions hold:

1. $\{R^*(\alpha') | \alpha' \in \mathcal{A}[\mathcal{S}]\} = \mathcal{A}[\mathcal{T}]$.
2. $\{R^*(\alpha') | \alpha' \in \mathcal{A}_\square[\mathcal{S}]\} = \mathcal{A}_\square[\mathcal{T}]$.
3. For all $\alpha' \in \mathcal{A}[\mathcal{S}]$, α' maps cleanly to $R^*(\alpha')$.

Definition. We say that $\mathcal{T}$ *follows* $\mathcal{S}$ (under R), and we write $\mathcal{T} \dashv_R \mathcal{S}$ if $\alpha' \to^{\mathcal{S}} \beta'$, for some $\alpha', \beta' \in \mathcal{A}[\mathcal{S}]$, implies that $R^*(\alpha') \to^{\mathcal{T}} R^*(\beta')$.

Definition. We say that $\mathcal{S}$ *models* $\mathcal{T}$ (under R), and we write $\mathcal{S} \models_R \mathcal{T}$, if for every $\alpha \in \mathcal{A}[\mathcal{T}]$, there exists $\Pi \subset \mathcal{A}[\mathcal{S}]$ where $R^*(\alpha') = \alpha$ for all $\alpha' \in \Pi$, such that, for every $\beta \in \mathcal{A}[\mathcal{T}]$ where $\alpha \to^{\mathcal{T}} \beta$, (1) for every $\alpha' \in \Pi$ there exists $\beta' \in \mathcal{A}[\mathcal{S}]$ where $R^*(\beta') = \beta$ and $\alpha' \to^{\mathcal{S}} \beta'$, and (2) for every $\alpha'' \in \mathcal{A}[\mathcal{S}]$ where $\alpha'' \to^{\mathcal{S}} \beta'$, $\beta' \in \mathcal{A}[\mathcal{S}]$, $R^*(\alpha'') = \alpha$, and $R^*(\beta') = \beta$, there exists $\alpha' \in \Pi$ such that $\alpha' \to^{\mathcal{S}} \alpha''$.

[1]Note that R^* is a total function since every assembly of S represents *some* assembly of T; the functions R and α are partial to allow undefined points to represent empty space.

The previous definition essentially specifies that every time $\mathcal{S}$ simulates an assembly $\alpha \in \mathcal{A}[\mathcal{T}]$, there must be at least one valid growth path in $\mathcal{S}$ for each of the possible next steps that $\mathcal{T}$ could make from α which results in an assembly in $\mathcal{S}$ that maps to that next step.

Definition. We say that $\mathcal{S}$ *simulates* $\mathcal{T}$ (under R) if $\mathcal{S} \Leftrightarrow_R \mathcal{T}$ (equivalent productions), $\mathcal{T} \dashv_R \mathcal{S}$ and $\mathcal{S} \models_R \mathcal{T}$ (equivalent dynamics).

3.2.2 Intrinsic universality

Now that we have a formal definition of what it means for one tile system to simulate another, we can proceed to formally define the concept of intrinsic universality, i.e., when there is one general-purpose tile set that can be appropriately programmed to simulate any other tile system from a specified class of tile systems.

Let REPR denote the set of all supertile representation functions (i.e., m-block supertile representation functions for some $m \in \mathbb{Z}^+$). Define $\mathfrak{C}$ to be a class of tile assembly systems, and let U be a tile set. Note that each element of $\mathfrak{C}$, REPR, and $\mathcal{A}^U_{<\infty}$ is a finite object, hence encoding and decoding of simulated and simulator assemblies can be represented in a suitable format for computation in some formal system such as Turing machines.

Definition. We say U is *intrinsically universal* for $\mathfrak{C}$ *at temperature* $\tau' \in \mathbb{Z}^+$ if there are computable functions $\mathcal{R} : \mathfrak{C} \to \mathsf{REPR}$ and $S : \mathfrak{C} \to \mathcal{A}^U_{<\infty}$ such that, for each $\mathcal{T} = (T, \sigma, \tau) \in \mathfrak{C}$, there is a constant $m \in \mathbb{N}$ such that, letting $R = \mathcal{R}(\mathcal{T})$, $\sigma_{\mathcal{T}} = S(\mathcal{T})$, and $\mathcal{U}_{\mathcal{T}} = (U, \sigma_{\mathcal{T}}, \tau')$, $\mathcal{U}_{\mathcal{T}}$ simulates $\mathcal{T}$ at scale m and using supertile representation function R.

That is, $\mathcal{R}(\mathcal{T})$ outputs a representation function that interprets assemblies of $\mathcal{U}_{\mathcal{T}}$ as assemblies of $\mathcal{T}$, and $S(\mathcal{T})$ outputs the seed assembly used to program tiles from U to represent the seed assembly of $\mathcal{T}$.

Definition. We say that U is *intrinsically universal* for $\mathfrak{C}$ if it is intrinsically universal for $\mathfrak{C}$ at some temperature $\tau' \in Z^+$.

Definition. We say that $\mathfrak{C}$ is intrinsically universal if there exists some U that is intrinsically universal for $\mathfrak{C}$ and for every $\mathcal{T} \in \mathfrak{C}$ and $\mathcal{U}_\mathcal{T}$ which simulates it, $\mathcal{U}_\mathcal{T} \in \mathfrak{C}$.

3.3 The directed aTAM is not intrinsically universal

Let $\mathfrak{D}$ represent the class of all tile assembly systems within the aTAM which are directed.

Theorem 1. $\mathfrak{D}$ is not intrinsically universal.

Theorem 1 states that there exists no aTAM tile set U such that, for any directed aTAM tile assembly system $\mathcal{D} \in \mathfrak{D}$, where $\mathcal{D} = (T, \sigma, \tau)$, there exists a directed aTAM system $\mathcal{U}_\mathcal{D} \in \mathfrak{D}$, where $\mathcal{U}_\mathcal{D} = (U, \sigma_\mathcal{D}, \tau')$, scale factor $m \in \mathbb{N}$, and representation function $R : B_m^U \to T$, such that $\mathcal{U}_\mathcal{D}$ simulates $\mathcal{D}$ under m-block representation function R at scale factor m. Essentially, there exists no "universal" tile set such that for any directed aTAM system, that tile set can be configured in a simulating system which simulates the original and is itself directed too.

Our proof of Theorem 1 will be by contradiction. Therefore, assume that such a universal tile set U, which can be used to simulate any directed system while using a directed system, exists. Given that U, we define an aTAM system $\mathcal{T} = (T, \sigma, 2)$ which is directed and forms an infinite terminal assembly, explain the growth of $\mathcal{T}$, and verify that it is directed. We provide a high-level overview of $\mathcal{T}$ in Section 3.4. We then show why there exists no directed aTAM system $\mathcal{S} = (U, \sigma_\mathcal{T}, \tau')$ which simulates $\mathcal{T}$. Section 3.5 contains a very high-level overview of that proof. Full details of $\mathcal{T}$ can be found in Section 3.6, and for the impossibility proof in Section 3.7.

3.4 Overview of the directed aTAM system $\mathcal{T}$

At the highest level, $\mathcal{T}$ self-assembles an infinite structure, starting from a single seed tile placed at the origin, and growing from left to right. In well-defined intervals, as the assembly grows eastward it initiates upward growths, an infinite series of sets of three "modules"

which are subassemblies able to grow almost entirely independently of each other once the main horizontal growing structure has placed the tiles which serve as the "input" for the growth of each. The aTAM is computationally universal [39], and in fact it is quite straightforward to design a tile assembly system which simulates the computation of an arbitrary Turing machine M (e.g. [32, 26]) by growing rows of tiles, one above the other, where each row represents the full configuration of M at a given time step (i.e. the tape contents, read/write head location, and state) in the values of the glues encoded on their north sides, and the row immediately above it represents the full configuration of M at the next time step (by designing the tile types appropriately so that the only tiles which can attach above a given row ensure that the new northern glue above a position which just had the read/write head encodes the value that would have been output given the state of M and the cell's previous value, and depending on the direction the head would have moved, either the tile representing the cell to the left or write would have a glue encoding the new state of M and the current value of that cell). To provide a logically infinite tape, the tiles can be designed to grow rows "on demand" by extending a row by one tile each time the simulated read/write head attempts to move past the end of the currently represented row.

The three modules which grow upward are logically grouped so that there is one of each type in a set. These three modules are designed so that they simulate three computations which require asymptotically differing space resources. As each set is initiated with inputs of increasing values, and as the assembly grows infinitely to the right, those space requirements ensure that the smallest module cannot perform the computations of the larger two, and the mid-sized module cannot perform the computations of the largest. The computations carried out by each set of grouped modules as well as the geometries to which they are each constrained are carefully designed such that two of the modules are necessarily completely "ignorant" of the eventual outputs of the others. However, these two modules are designed so that after performing their computations, they grow assemblies represent-

ing bit strings corresponding to the outputs of their computations in locations across a one tile wide gap from each other, which we call the `bitAlley`. In locations where output bits of the two computations match, tiles attach between tiles for those bit positions. The third module independently computes the results of the computations of both other modules and if and only if there will be no matching bits between them, it grows an assembly which is a single tile wide path down through the `bitAlley` (thus it is guaranteed not to crash into any tiles in the `bitAlley`, regardless of the ordering of tile attachments). As the overall assembly grows further right, the inputs to the modules increase and the computations simulated by the modules require more resources and the `bitAlley`s become arbitrarily long. We are able to first show that $\mathcal{T}$ is directed, and then that no simulating system can be built using the tiles of a universal simulating tile set U and be itself directed. This is because any such directed simulator is forced by the dynamics of correct simulation, the mutual obfuscation of computations across modules, and geometric constraints, to effectively create bottlenecks which do not allow enough information to be transmitted to the growing assembly for correct growth and therefore simulation. The intuition is that the simulator has to make "guesses" about when it may need to place tiles which cooperate across a `bitAlley` (i.e. glues from the tiles on both sides of the gap are required to allow the attachment of one between them) which, due to the fact that space cannot be reused in the aTAM, doom it to failure. Furthermore, these guesses are required not by nondeterminism about which tiles can be placed in locations by $\mathcal{T}$, since after all $\mathcal{T}$ is directed, but rather due to the ordering of arrival of tiles - the particular assembly sequence which may be followed.

3.4.1 Overview of modules of $\mathcal{T}$

Figure 3.1 shows a schematic depiction of a portion of the terminal assembly of $\mathcal{T}$. We now give a very high-level description of each of the main modules, and full details can be found in Section 3.6.

Beginning from the seed, the module which grows horizontally and initiates growth of

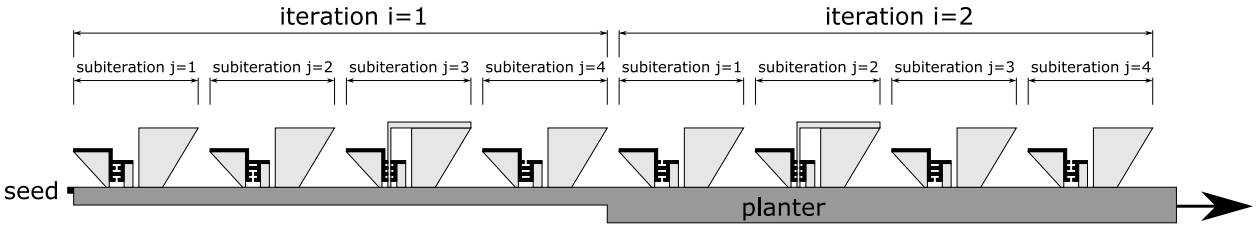


Figure 3.1: A high-level schematic depiction of a portion of the infinite assembly produced by a directed aTAM system $\mathcal{T}$ which cannot be simulated by any directed universal simulator.

sets of modules to its north is called the `planter`. The `planter` grows in a zig-zag, up and down manner, growing one column at a time. Essentially, its job is to manage a set of nested counters, whose values are used to (1) determine the correct spacing between the modules to the `planter`'s north, and (2) serve as input to those modules. The outermost of the nested counters counts $0 < i < \infty$, with each i being what we call an *iteration*. For each value of i that it counts, it holds that counter constant while it increments an inner counter from 0 to (approximately) 2^i. For each value of j it initiates the growth of what we call a *subiteration*. See Figure 3.2 for a high-level overview of one type of subiteration. For each subiteration, the `planter` counts out a sequence of spacing columns (i.e. columns whose sole purpose is to put horizontal space between modules) while also computing the value $\log(i)$ and then rotating the values of the bits representing $\log(i)$ upward so that they are encoded in a row of glues on the north sides of the northern tiles of the `planter`[2]. From these, a `left` module begins growth. This module performs a stacked up series of i Turing machine simulations on progressively increasing input values, with each simulation outputting a 0 (for a rejecting computation) or a 1 (accepting). At the top of the stack of computations, the string of output bits is rotated to the right and then grown downward to the right of the `left` module. Once that growth reaches a specially marked location, the values of those bits are rotated to the right where they are presented as the eastern glues of the tiles forming the `bitAlley`. (See Figure 3.3 for a depiction of a southern portion of a `bitAlley`.)

[2]Note that throughout this paper, log means $\log_2$, and we use the shorthand $\log(i)$ to mean $\lceil \log(i) \rceil$.

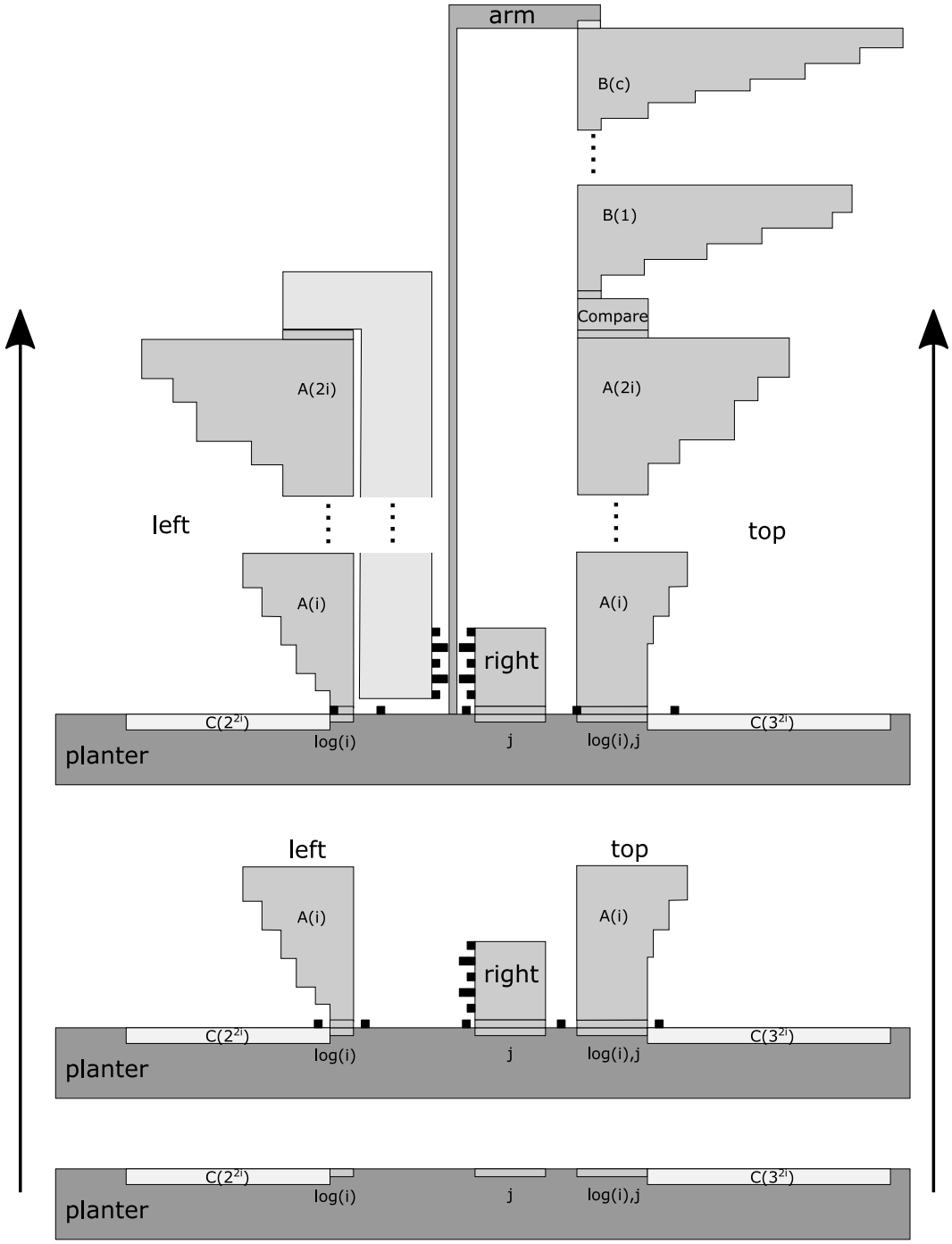


Figure 3.2: A high-level schematic depiction of one possible ordering of growth of the modules of an empty subiteration. (Bottom) The `planter` lays out the inputs for the modules at the necessary spacings to prevent them from colliding, (Second) The `left`, `right`, and `top` modules begin growth, (Top) Once the `top` completes it initiates the growth of the `arm` which grows down through the `bitAlley`. Note that an `arm` only grows in the `bitAlley` of an empty subiteration, unlike the `bitAlley` in Figure 3.3 which shows tiles cooperatively binding across the `bitAlley` of a non-empty subiteration. Also, empty subiterations occur exponentially more rarely than non-empty ones.

After growing a few spacing columns past the initiation point of the `left` module, the `planter` rotates the value of j to its north side to initiate growth of a `right` module. This module simply rotates the values of the bits of j to the left so they can be presented across the `bitAlley` from the bits output by the `left`. Note that as the iteration number i increases, so does the number of bits presented on each side of the `bitAlley`, as the `left` performs (approximately) i Turing machine simulations, and `right` actually receives the value of j in binary padded with 0's as necessary to be the same length.

The final module to be initiated by the `planter` in each subiteration is the `top` module. This module receives as input both the values $\log(i)$ and j. It first performs the same i simulations that the `left` performs, generating the same output bits. It then compares those bits to the bits of j to determine if there are any locations where the bits are the same. If there are, then in the `bitAlley` there will be tiles which attach between them across the gap in those locations, and the `top` module halts its growth (in this subiteration). It is guaranteed that in exactly one subiteration of each iteration that there will be no matching bits, since each subiteration performs the same `left` computations on the same input and there is a unique subiteration for every possible bit string of length i, exactly one of which can be the complement of `left`'s output on that input. In this special subiteration of the iteration, which we call the *empty* subiteration (because the `bitAlley` will be empty of tiles cooperating across the gap), the `top` performs a new set of computations to determine which of a large number (relative to the number of tile types in the claimed universal simulator U) of `arm` modules to grow. The `arm` module grows over to a position directly above the `bitAlley`, then grows a single tile wide column of tiles down through the `bitAlley` until it crashes into the `planter`, with the specific type of tile used for the `arm` determined by the final computations performed by the `top` module. This completes the growth of a subiteration, and the growth of subiterations and iterations occurs for infinite numbers of each.

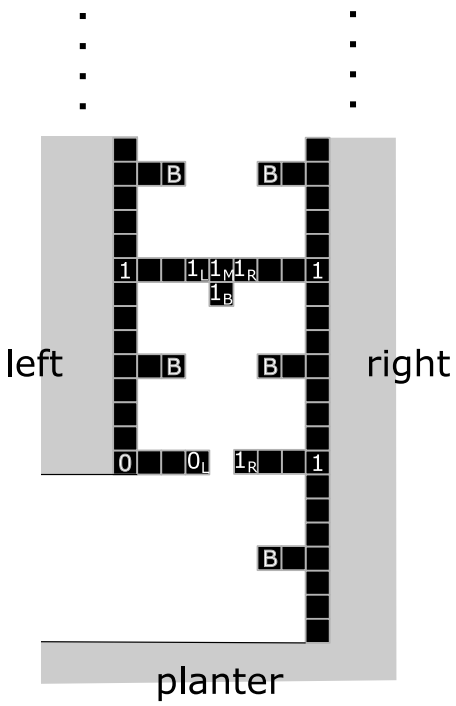


Figure 3.3: Example `bitAlley` portion between `left` and `right` modules of a non-empty subiteration.

3.4.2 Directedness of $\mathcal{T}$

The system $\mathcal{T}$ is directed because there are no locations where tiles of multiple types might be placed during different assembly sequences, and this is ensured by carefully dictating the growth of each module (all grow in zig-zag manners), and the amount of space required for each is carefully computed and accounted for by the `planter` so none of them can collide. Finally, the `arm` will only grow in empty subiterations, which can be assured by the `top` module performing the computations of `left` and comparing the output bits to j, so it will never collide with tiles in the `bitAlley`. Thus, despite the fact that there are an infinite number of unique assembly sequences in $\mathcal{T}$, they all result in the exact same terminal assembly in the limit.

3.5 Overview of impossibility of simulation

In this section we provide a high-level overview of the proof that $\mathcal{S}$ does not simulate $\mathcal{T}$. More details can be found in Section 3.7.

The general idea behind the proof that $\mathcal{S}$ cannot simulate $\mathcal{T}$ is based around creating a situation in $\mathcal{T}$ where there is a one tile wide gap between two tiles such that, depending on their types, they may or may not cooperate to place a tile in between them (i.e. a tile may bind using one glue from each of them). However, if and only if all of these tiles in the `bitAlley` do not cooperate to place a tile between them, another assembly will grow between them without binding to either of their glues. In $\mathcal{T}$, the gap is exactly one tile wide and so is the assembly that may grow down through it. Since we are proving by contradiction, assume that such an $\mathcal{S}$ exists and that it has tile set U with size $|U| = t$. We design $\mathcal{T}$ such that the number of unique `arm` module tiles (which are the ones that grow between the two tiles if they do not cooperate) is exponentially larger than t. This forces the simulation scale factor m used by $\mathcal{S}$ to be larger than 1 because any macrotile created from tiles in U must have enough tiles to uniquely identify any of the tile types in $\mathcal{T}$. Then we also note that geometrically, the only way to get two tiles to cooperate to place a tile in between them is for them to grow to positions with less than or equal to a single tile wide gap between them, which is not enough room for the macrotile of an `arm` module, with $m > 1$, to pass through if necessary. While the general idea seems simple, first, care must be taken in designing $\mathcal{T}$ so that an `arm` module will be grown if and only if the tiles will not cooperate across the gap, with no chance for a disagreement and collision since $\mathcal{T}$ must be directed, so the portion of the assembly which initiates the growth of the `arm` must be able to compute the tiles which will appear across the gap from each other. Then, it must be shown that $\mathcal{S}$ is forced to grow all the way to a single tile wide gap even when cooperation won't be necessary, thus blocking the `arm`. The main difficulties arise with the realization that the simulating system could attempt to compute in advance if cooperation will occur and, if so, grow to the one tile wide gap which allows for cooperation, but if

not, stop growth short of that to leave enough room for the `arm` module to grow through. The resulting complexity of $\mathcal{T}$ arises from the need to create a system which is "confusing" enough for the simulator that the modules growing the macrotiles representing the tiles which may cooperate across the gap are unable to pre-compute the answer to whether or not cooperation will be necessary. Essentially, the fact that $\mathcal{S}$ cannot both cooperate and/or grow a full tile-representing assembly through a single tile wide gap dooms it to failure, but extensive machinery is required to force the situation.

A key tool in the proof is that in an arbitrary subiteration j of an arbitrary iteration i, the output of the `left` module is impossible to compute from within either the `planter` or the `right` modules, and the output of the `right` is impossible to compute from within the `left`. The reason for this is that (1) the Turing machines being simulated within the `left` modules are deciding languages which cannot be recognized in infinitely often best-case space complexity [33] which is greater than the space resources available to the `planter` and `right` modules, and thus the outputs of `left` modules cannot be computed by them, and (2) the input j passed to the `right` module is asymptotically much greater in size than the amount of information which can be input to the `left` module through the only $\log(\log(i))$ macrotiles allowed in the bottom row of the `left` module to encode the value $\log(i)$, making it unable to get asymptotically more than a log size chunk of the `right` module's input. It is also important to note the languages being decided within the `left` are recognized in almost everywhere worst case space complexity which is accounted for by the spacing columns of the `planter`, guaranteeing that for all but a finite number of computations, the `left` will be able to successfully complete its computations. It will prematurely abort any computations which attempt to run beyond those space bounds, but since there are guaranteed to be only a finite number of those, the goals of the construction and correctness of the proof aren't compromised. It is important that these essentially arbitrarily tight bounds on the space complexities of languages is shown to be possible by Theorem 4.1 of [33], which allows for the computations embedded within the modules to

be designed with great precision. In a similar manner, the computations performed by the upper portion of the `top` module require space complexity greater that that available to either the `planter` or `left` of the same subiteration. We note that Lemma 9 is instrumental in proving the above facts, and is also an important tool which can be used in future simulation-based results in the aTAM, as it proves that an assembly performing a simulation of a system growing in a zig-zag manner, despite its arbitrarily large (but constant) scale factor, has asymptotically no greater space resources available than the orignal system. The technical tools we have developed for this proof, as well as the incorporation of results from complexity theory allowing for precisely defined languages in terms of space complexity, provide a host of new construction and proof techniques which we feel will be useful for a variety of future results.

To prevent the simulator from being seeded with answers to the necessary computations, the assembly of $\mathcal{T}$ must grow infinitely many iterations and subiterations. To prevent other types of "cheating", rather than having potential locations of cooperation across a single gap between two tiles, the `bitAlley` becomes arbitrarily long, between an arbitrarily large set of pairs of tiles. To prove all of the necessary properties of the simulator $\mathcal{S}$ requires many more details and the use of several additional technical lemmas which may possibly be of independent interest and utility. Please see Section 3.7 and Section 3.8 for full details.

3.6 Details of the directed system $\mathcal{T}$

In this section, we provide details of the construction of $\mathcal{T}$ as well as explaining its growth and verifying that $\mathcal{T} \in \mathfrak{D}$, i.e. that $\mathcal{T}$ is directed.

3.6.1 Languages and Turing machines used

The decidable languages and the Turing machines which decide them and are simulated within the `left` and `top` modules of $\mathcal{T}$ are defined as follows.

Let $a = 2^{2t}$ where $t = |U|$ is the size of the simulator's tile set. Also, let $s : \{0,1\}^n \times \mathbb{N} \to \{0,1\}^*$ be defined by $s(x,n) = x \cdot 0^n$ where $\cdot$ represents concatenation. That is s extends a string by adding n zeros to the end of the string.

Let $L_A \subset \mathbb{N}$ be a decidable language such that L_A can be decided in almost everywhere worst-case space complexity 2^n and L_A cannot be recognized in infinitely often best-case space complexity $(1/2)2^n$, i.e. for all but finitely many n, L_A requires space greater than $(1/2)2^n$. Let A be a deterministic Turing machine which decides L_A within space 2^n almost everywhere, i.e. for all but finitely many n. Note that such an L_A is guaranteed to exist by Theorem 4.1 of [33].

Let A^+ be a Turing machine which, on input $\log(x)$ of length n, does the following. For each $0 \leq i < 2^n$, A^+ simulates $A(s(x,i))$ and records a 0 if A rejects, and a 1 if it accepts. Furthermore, while A^+ is performing any simulation of $A(z)$, it bounds the space used by the computation and if the machine attempts to use $2^{|z|}+1$ unique tape cells, it halts that simulation and records a 0 for it. Furthermore, it also bounds the time used and if it attempts to use more than $2^{2^{|z|}}$ time steps, it halts that simulation and records a 0 for it. Once the full series of simulations of A have completed, A^+ halts with a binary sequence of length 2^n on its tape, representing the outputs of each of the 2^n computations. Furthermore, A^+ uses a one-way-infinite-to-the-left tape.

Note that given input $\log(x)$ of length n, the maximum amount of space used by A^+ will be used during its computation of $A(2^{s(x,2^n)})$, which is bounded by A^+ at $2^{2|x|}+1$.

Furthermore, by the fact that L_A can be decided in almost everywhere worst-case space complexity 2^n, and that for any constant c there exists some i' such that for all $i > i'$, $2i > ci'$, then for all computations $A(i)$ beyond those for a constant number of values, $A(i)$ is guaranteed to halt without using space greater than $2^{|i|}$ or more than $2^{2^{|i|}}$ time steps, and thus all computations of A on inputs greater than that will

successfully complete without being halted by A^+.

Let $L_B \subset \mathbb{N}$ be a decidable language such that L_B can be decided in almost everywhere worst-case space complexity 3^n and L_B cannot be recognized in infinitely often best-case space complexity $(1/2)3^n$, i.e. for all but finitely many n, L_B requires space greater than $(1/2)3^n$. Let B be a deterministic Turing machine which decides L_B within space 3^n almost everywhere, i.e. for all but finitely many n. Note that such an L_B is guaranteed to exist by Theorem 4.1 of [33].

Let R be a Turing machine which uses a one-way-infinite-to-the-right tape and takes as input two binary strings, i and j both of length n, and performs the following computations:

> R first runs $A^+(\log(i))$ and compares the resulting i-bit string to j.
>
> If those bit strings match on at least one bit, R halts.
>
> Otherwise (i.e. when they differ on every bit), for $0 \leq k < a - 1$, R simulates $B(s(i,k))$ and records a 0 if B rejects k, and a 1 if it accepts. Furthermore, while R is performing any simulation of $B(s(i,k))$, it bounds the space used by the computation and if $B(k)$ attempts to use $3^{2|i|}+1$ unique tape cells (i.e. space equal to $3^{2|i|}+1$), it halts that simulation and records a 0 for it.
>
> Once the full series of simulations of B have completed, R halts with a binary sequence of length a on its tape, representing the outputs of each of the a computations.
>
> Note that given inputs i and j, the maximum amount of tape cells used by R will be used during the computation of $B(s(i,a))$ which requires space $3^{(|i|+a-1)}$ for all but finitely many i, which is bounded by R at $3^{(2|i|)}$.
>
> Furthermore, by the facts that $L_B \in \mathsf{DSPACE}(3^n)$, and that for any constant c there exists some i' such that for all $i > i'$, $i^2 > ci$, then for all computations $B(i)$ beyond those for a constant number of values, $B(i)$ is guaranteed to halt

without using space greater than $3^{2|i|}$, and thus all computations of B on inputs greater than that will successfully complete without being halted by R.

Let $\mathcal{T} = (T, \sigma, 2)$ be the directed aTAM system which self-assembles the infinite shape sketched in Figure 3.1. We will discuss the assembly it produces in a modular way. The seed σ consists of a single tile placed at the origin. From the east side of the seed, the "`planter`" module forms.

3.6.2 `planter`

The `planter` module does the following (and is conceptually somewhat similar to the "`planter`" module discussed in Section 4.5 of [26]). It is in general a log-height binary counter (i.e. a binary counter which represents consecutive numbers by bit strings in consecutive columns, and each column having height equivalent to the number of bits in the number being represented by the counter) which enumerates the positive integers, counting from 1 to ∞. For each positive integer $i \in \mathbb{Z}^+$, it initiates the growth of an *iteration*, which consists of 2^i *subiterations*, each of which initiates growth of a set of modules which grow from the north side of the `planter`. The function of the `planter` is to correctly space out those modules by putting them at well-defined locations, as well as to provide input to each via north-facing glues of rows of tiles which initiate the growth of each module.

To do this, the `planter` actually contains a series of embedded counters, with the counter for the iterations, i, being the "outer" counter. For each value of i, before the `planter` increments i again, it begins a nested counter j which it iterates over the values $0 \leq j < 2^{2^{\log(i)}} = 2^i$[3]. (During the columns where the value of i is not incremented, its bit values are simply passed forward, i.e. to the right, unchanged.) For each value of j it will initiate growth of a subiteration by incorporating additional counters used to guarantee correct spacing of modules. Therefore, there are also counters nested within each subiter-

[3]Here and throughout the paper, we will use the shorthand $\log(x)$ to mean $\lceil \log(x) \rceil$ and also use the shorthand $x = 2^{\log(x)}$ despite the fact that $x \leq 2^{\lceil \log(x) \rceil}$, as it will not impact the correctness of our arguments.

ation j. The first of these is a counter which counts from 0 to 2^{2i}, by first computing 2^{2i} (simply by starting from the binary string 1 and while counting from 0 to i adds another bit position for each count, e.g. 10, 100, etc.), and then starting a counter at 0 which increments each column while passing the value 2^{2i} through to the right and checking each counted value until it matches 2^{2i} at which point it halts. The horizontal distance grown by that counter is used to create enough space for the next module whose growth will be initiated on the north side of the `planter` at this point.

While continuing to grow to the east, the `planter` now computes the value $\log(i)$ (by simply counting the length of the binary representation of i) and rotates a copy of the value $\log(i)$, whose length is $\log(\log(i))$, northward so that after growing another $\log(\log(i))$ columns, the binary value of $\log(i)$ is represented by the north-facing glues of $\log(\log(i))$ tiles. This binary representation of $\log(i)$ will serve as the initiation point for the growth of a `left` module, to be discussed later. In addition the `planter` exposes a τ strength north-facing glue two tiles to the west of the beginning of the north-facing glues which represent the binary value of $\log(i)$ and also exposes a τ north-facing glue two tiles to the east of the end of the glues which represent the binary value of $\log(i)$. These two glues allow for the growth of two tiles which attach via north and south glues which we call a `bumper`. Also, in locations not specifically mentioned, the northern glues of the northernmost tiles of the `planter` are . The `planter` now grows an additional $2^i + \log(i) + 11$ columns to the right, at which point it rotates a copy of the binary value of j to the north. This binary representation of j will serve as the initiation location for a `right` module, also to be discussed below. Next, the `planter` grows another 7 spacing columns to the right, and then rotates copies of the binary representations of both i and j to the north, so that the value i,j can serve as the initiation for a `top` module. Similar to the initiation point for the `left` module, we place τ strength glues around the initiation points of the `right` and `top` initiation point which allow for the growth of `bumper`s.

The last bit of growth for the `planter` in a subiteration is to grow the spacing rows

which are necessary for the `top` module (since for its growth above the `planter` it will grow both upward and to the right). For this, in a manner analogous to the way it grew the 2^{2i} spacing rows for the `left` module, it now grows 3^{2i} spacing rows. At this point, the `planter`'s growth in relation to the jth subiteration of the ith iteration is complete. It now increments the value of j (assuming it is $< 2^i$, or specifically $< 2^{2^{\log(i)}}$), and grows the necessary columns for the next subiteration, or, if j now equals 2^i, it resets j to 0 and increments the value of i to begin a new iteration. This occurs for infinitely many iterations.

The important features of the `planter` are that (1) it initiates the appropriate growth to allow the modules of each subiteration to grow independently of the `planter` once it has written the necessary input values for those modules, (2) that in between the inputs for the modules it creates a necessary amount of spacing columns to ensure that no modules will collide with each other due to the space complexities of the languages whose accepting Turing machines are being simulated by each (to be explained below), (3) all growth is performed in an up and down zig-zag manner, with each column completely growing before the column to its right begins growth either at the top or bottom of the column (depending on whether the column is zigging or zagging), and (4) the number of tiles used in each column, i.e. the height of the `planter`, is never more than the minimum needed for each position to represent a single bit of each counter value being propagated through. Since the largest value of any such counter is 3^{2i}, that means that the maximum height of the counter during any iteration i is $\log(3^{2i}) = O(2i)$. See Figure 3.2 for a sketch of the formation of an iteration.

3.6.3 `left`

For all values i, the `left` module receives the input $\log(i)$, whose width is $\log(\log(i))$. The `left` module performs a simulation of $A^+(\log(i))$ by growing a zig-zag Turing machine which begins with a tape whose width is the width of its input plus one, with the extra cell representing a specially marked `blank` tape cell denoting the current end of the tape. When, and only when, A^+ tries to access the leftmost tape cell, that cell is interpreted as

a regular `blank` and the end marker is moved one position to the left to a tile which grows that column one position to the left. In this way, the simulation of A^+ uses rows whose width are the same as the number of tape cells used by $A^+ + 1$. Figure 3.2 shows how the system $\mathcal{T}$ simulates the computations that compose A^+. The system simulates the computation A^+ by passing each computation $A(j)$ which composes A^+ three pieces of information via glues exposed on the north of the previous computation $A(j-1)$: 1) the input to the machine $A(j)$, 2) the outputs of the previous computations $A(k)$ for all $k < j$, and 3) the value $2^{\log(i)+1}$. In this way, the machine is able to simulate the computations which compose A^+, and halt on the last machine $A(i)$, and then write the outputs of all the machines along the north border of the machine $A(i)$ as shown in Figure 3.2. In addition, we embed a counter in the `left` module which counts the number of total time steps used by all machines. Since each computation in A^* can use at most 2^{2^i} time steps, the total time steps used by all machines is at most $i * 2^{2^i}$. Consequently, to store the number of time steps used, space $\log(i) + 2^i$ is needed to store the values of the counter. Note that we can embed this in the same cells which are used in the computation of A^+.

Once $A^+(\log(i))$ halts, with a bit string of length i (i.e. $2^{\log(i)}$) and the total number of time steps ts output to its north, those output bits are rotated up and to the right, three columns to the right beyond the right side of the computations below, and then a $\log(i) + 2^i$-width counter counts down starting from the value ts until it is a distance $8 \times 2^{\log(i)} + 4$ above the planter at which point the output bits of $A^+(\log(i))$ are rotated to the right, one at a time and with 7 tiles vertically between each. Figure 3.3 shows an example of the first two bits output by the `left` module. Notice that between each bit that is output there are seven tiles and a tile which is a distance of three tiles away from each output bit grows two tiles via $E - W$ glue attachment which we call a `bumper`.

Recall that for almost all x, $A^+(x)$ has worst case space complexity $2^{2|x|}$. Consequently, this means that for almost all iterations i the `left` module will need at most space 2^{2i}. Note that the `planter` was designed to space the modules so that in iteration i the `left`

module has 2^{2i} space available for use. Consequently, for almost all i, all of the computations performed in the `left` module will complete without prematurely halting.

3.6.4 `right`

The `right` module receives the input j and simply rotates those bits upward and to the left, and also spaces them out with 7 tiles in between each bit, along with 7 before the first bit, so that they are output on the west side of the `right` module across $7i$ rows (since j consists of i bits). An example of the first two bits output by the `right` module can be seen in Figure 3.3.

3.6.5 `top`

As soon as the `planter` completes the formation of the portion of its northern row encoding the input for the `top` module, namely the encoding of i and j near the eastern side of a subiteration, the `top` module is able to begin simulating the Turing machine R which is defined above. Recall that R first runs $A^+(\log(i))$. As shown in Figure 3.2 the `top` module first simulates the A^+ computation on input $\log(i)$ in the same manner as the `left` module with the exception that it simulates a TM which computes the A^+ computation using a one-way-infinite-to-the-right tape (rather than to the left). As the `top` module simulates A^+ on input i, it also propagates the value of j via glues.

Next, the Turing machine R compares the output of $A^+(i)$ to the value j, so we design the `top` module so that it mimics this behavior. Once `top` finishes it's simulation of A^+ on input i, it then compares the output of this computation to the value j. If they match on any bits, the growth of `top` terminates. However, if they differ in every single position, the `top` module simulates the series of computations $B(s(i,k))$ for $0 \leq k < |i|$ in a manner similar to it's simulation of the A^+ computation. After the `top` completes the simulation of the last machine $B(s(i,k))$, it outputs the a-bit string which is the output of the series of computations to its north. This initiates the growth of the `arm` module as shown in Figure 3.2.

Recall that for almost all x, $B(s(x,a))$ has worst case space complexity $3^{|x|+a}$. Consequently, this means that for almost all iterations i the `top` module will need at most space $3^{2|i|}$. Note that the `planter` was designed to space the modules so that in iteration i the `top` module has $3^{2|i|}$ space available for use. Consequently, for almost all i, all of the computations performed in the `top` module will complete.

3.6.6 `arm`

If `top` initiates the growth of the `arm` module, there are a possible 2^a different binary values written as output by the `top` computations, and the function of the first row of the `arm` is to grow to the west across those output bits so that when that completes it has selected one of 2^a possible types of `arm`s to grow. The arm module is essentially first a horizontal counter which grows leftward $7+j+4$ (to pass over the spacing columns between the `right` and `top` modules, the `right` module, and end up directly over the center position of the *bit alley* between the `left` and `right` modules. Once the `arm` reaches the end of its leftward growth, it then initiates a downward growing column from its leftmost column. This downward growing column consists of a single repeating tile type so that the column eventually crashes into the `planter` between the `left` and `right` modules of the subiteration. The tile type of the column is determined by the value written by the `top` module, and can be any of 2^a types.

3.6.7 `bitAlley`

Figure 3.3 shows a portion of possible assembly growth between the `left` and `right` modules of a subiteration, say the jth of iteration i. As noted in the above section, the counter which grows from the top of `left` causes the growth of tiles as shown in Figure 3.3 which depends on the output of $A^+(i)$. If the bit is a 0, the rightmost of those tiles is of type 0_L, else it's of type 1_L.

The height of `right` is $8i + 4$, and spaced out similarly to the bits on the east side of the `left` module, it has the bits of j presented on its west side. Also similarly to the oppo-

site side, from each bit a pair of tiles attach with the westernmost being of type 0_R or 1_R, depending on each bit value of j.

Since the bit strings exposed by `left` and `right` are aligned with each other, at each position where a bit on the left matches one on the right, a tile cooperatively binds to the two tiles, either 0_L and 0_R or 1_L and 1_R. Without loss of generality, assume such a matching bit position has value 0. Then, between the 0_L and 0_R which are at the same height and separated horizontally by a single space, a tile of type 0_M binds "across the gap." Finally, a tile of type 0_B attaches to the south of the 0_M tile and growth related to this bit is complete. Such attachments of 0_M, 0_B, 1_M, and 1_B tiles occur at each position where bits match across `left` and `right`, and at each position where they differ, the final tile attachments are the 0_L, 0_R, 1_L, and 1_R tiles.

Finally, recall that exactly in subiterations where the bits in every position of `left` and `right` differ, an `arm` grows, which results in a single-tile-wide column of tiles growing southward from the `arm`, between the 0_L, 0_R, 1_L, and 1_R tiles of the `left` and `right` modules, and crashing into the `planter`. Since this occurs if and only if every bit position differs, there is no possibility of a cooperatively placed 0_M or 1_M tile blocking the growth of this column.

3.6.8 Summary of computations

Module	Input	Computation	Space complexity	Also able to compute
`planter`	none	count, $\log(i)$	$O(i)$	none - insufficient space
`top`	j,i	$A^+(\log(i))$,$B(\log(i))$	$O(3^i)$	$A^+(\log(i))$, $R(i)$
`left`	$\log(i)$	$A^+(\log(i))$	$O(2^i)$	none - insufficient input
`right`	j	*none*	$O(i)$	none - insufficient space

Table 3.1: The computations performed by each module in subiteration i,j.

3.6.9 The system $\mathcal{T}$ is directed

Since the interesting components that compose our system are based off zig-zag systems which are clearly directed, the only potential sources of nondeterminism are 1) the modules

which perform computations using too much computational resources and crashing into each other, and 2) the arm growing from the `top` module causing a race condition to be created between the arm and a tile that is placed cooperatively in the bit-alley. The first situation is prevented from arising by the counters embedded in the `left` and `top` modules and the appropriate spacing provided by the `planter`. The second scenario cannot arise due to the fact that the `top` grows an `arm` if and only if the output of $A^+(\log(i))$ disagrees on all bits with j which means that no tile can be cooperatively placed in the `bitAlley`. Thus, $\mathcal{T}$ is a directed aTAM system.

3.7 Details of impossibility of simulation

The proof that $\mathcal{S}$ does not simulate $\mathcal{T}$ consists of two main portions, each geared toward showing that it is impossible for modules within a subiteration to receive and utilize any information about the output of the complex computations occurring in the `left` or `top` modules prior to those computations occurring, or outside of the modules performing those computations. Such information could potentially have allowed $\mathcal{S}$ to remain directed while accurately simulating $\mathcal{T}$, but instead the lack of such prior information and the chance to effectively utilize it leads to a contradiction that $\mathcal{S}$, and thus U, exists.

We first show that the probes that the `left` and `right` modules grow on the sides of the `bitAlley` of each subiteration must be grown, in at least an infinite number of iterations, so that there is nothing unique about those in the empty subiteration. Intuitively, this means that at least infinitely often the probes grown in subiterations must be ignorant of whether they will need to cooperate across the `bitAlley` and therefore "attempt" to grow to positions that leave no more than a single tile wide gap in the `bitAlley` to allow for correct simulation in situations where cooperation will be required across the `bitAlley`.

The second main point shows that, given the infinite set of iterations just proven to exist, it must be the case that the bottlenecks created across the `bitAlley` are either not sufficient to allow the necessary `arm` modules to grow or conversely for the necessary co-

operation to occur across the `bitAlley`. Part of this relies on showing that the only way the necessary variety of possible `arm` modules could grow is for the probes which partially block the `bitAlley` to encode information about at least which subset of possible `arm` modules the actual `arm` module to be grown will be selected from, in advance of the `top` module completing its computations which determine the `arm` type. This would allow the probe tiles to assist in the formation of the `arm` modules, but is shown to be impossible.

The proofs of each of these portions rely upon the fact that if either of those types of information were provided in advance to the growing modules, then it would be possible to construct Turing machines which simulate the assembly of $\mathcal{S}$, inspect the subassemblies thus created, and utilize that information to solve instances of computations which are known to require more space resources than such Turing machines would be using, providing the necessary contradictions. Note the tight reliance upon the computational complexities of the corresponding decidable languages and the ability to use the tools we have developed to quantify and bound the computational resources available to the subassemblies performing the computations. Many of these tools can be found as technical lemmas, with associated proofs, in Section 3.8.

3.7.1 Empty subiterations cannot be uniquely marked in advance

In this section, we show that it is impossible for the `left` and `right` modules of empty subiterations to "cheat" by not growing valid complementary pairs of probes, and we show how that prevents $\mathcal{S}$ from successfully simulating $\mathcal{T}$. We first define some useful terms and show some properties which must be true of the assembly produced by $\mathcal{S}$.

Definition. Given a subassembly $\alpha \sqsubset \alpha_U$ which represents the single tile wide vertical portion of an `arm` in $\mathcal{T}$, let $\alpha_r \sqsubset \alpha$ be the largest subassembly of α such that, below some initial subassembly $\alpha_0 \sqsubset \alpha$ which occurs at the top of α, α consists only of repeated, nonoverlapping copies of α_r, one immediately below the other, and the topmost immediately below α_0. Note that the bottommost copy of α_r may be truncated, and if

there is no such repeating portion of α then dom $(\alpha_r) = \varnothing$. We say that the *shape* of α is dom $(\alpha_0) \cup$ dom (α'_r) where $\alpha'_r \sqsubset \alpha$ is the topmost copy of α_r.

Claim 1. Let S be the set of all unique shapes of `arm`s in α_U. Then $n_{arms} = |S|$ depends only upon the number of tile types in U, $t = |U|$, and the scale factor of the simulation, m.

Proof. To prove Claim 1, we utilize Lemma 5.10, the Closed rectangular window movie lemma (CRWML), of [21]. Note that each `arm` is a single tile wide in $\mathcal{T}$, and that in any empty subiteration it is possible for the `top` and `arm` to grow before either the `left` or `right` have even begun growth. In $\mathcal{S}$ it must be possible to simulate such an assembly sequence, and that means that during the growth of the macrotiles in $\mathcal{S}$ representing the `arm` tiles of $\mathcal{T}$ in such an assembly, there must be no tiles beyond a single macrotile to either their left or right sides (as those locations must map to empty space and therefore tiles in those regions are considered fuzz, which is not allowed to extend further than a single macrotile away from some macrotile which maps to a nonempty location). In such an assembly, the maximum width of any horizontal cut across such an `arm` subassembly in $\mathcal{S}$ is $\leq 3m$, or the width of 3 macrotiles. Let $h = 2(3m!(4t)^{3m})$ be a constant to be explained below. To characterize all possible `arm` shapes, we iterate over every possible configuration of tiles from U which form any subset of a $3m$-tile-wide line, and for each simulate all assembly sequences which place tiles only to the south of that line, until they reach a distance $2h$ or they produce an assembly to which no additional tiles can attach below that configuration but which doesn't reach a distance of $2h$. (We don't need to consider any which grow above the line because if that growth eventually influences the assembly below, it must do so via a path containing a tile which crosses that line, and the configuration consisting of the current configuration plus that tile will also be simulated.) We save the shape s of the `arm` created by that configuration in S provided the following hold: (1) all assemblies produced from all possible assembly sequences starting from that configuration grew only within the 3 macrotile wide space directly below it, and all place the same tiles at all locations, and (2) s is not already in S.

A window movie (as defined in [21, 29]), is a set containing the locations, types, and orderings of arrival of glues along a cut across an assembly. If we consider windows (i.e. boxes which separate a grid graph into interior and exterior portions) whose top edges cut directly across `arm` subassemblies and whose other edges do not pass through any portion of an assembly, we note that the number of window movies is $\leq 3m!(4t)^{3m}$ (hence our previous choice of h). If we inspect all `arm`s whose shapes were saved into S, we first note that the number which belong to `arm`s which did not reach distance $2h$ cannot be larger than the number of ways to tile the $3m \times 2h$ region using only t tile types, which is a constant dependent only upon t and m. We then inspect each whose `arm` did grow to a distance of $2h$, and we note that by the pigeonhole principle, any subassembly representing an `arm` which longer than h must have two windows $w \neq w'$ such that their window movies are the same (i.e. the same glues arrive along those cuts in the same locations and orders). If we let α and β of the CRWML both be such an `arm` assembly, and the windows be w and w', the CRWML tells us that the subassembly of the `arm` between the first and second identical window movie locations could grow again after the second, and this can be applied an arbitrary number of times to show that that subassembly can be repeated indefinitely (unless blocked by some other subassembly). (This repeating subassembly corresponds to the α_r of the definition of the shape of an `arm`.) Because the number of window movies possible is determined only by t and m, so is the number of repeating subassemblies corresponding to α_r in the definition of the shape of an `arm`. Since the prefix α_0 is also bounded by height h, there are a fixed number of possible prefixes, again only dependent upon t and m, so the total number of unique shapes is also bounded by a constant dependent only upon t and m. □

Definition. The set of probes which grow on the left side of the `right` module of each subiteration can be compared with every `arm` shape to generate a subset of `arm` shapes which would not be geometrically blocked by those probes. That is, assuming that the macrotiles representing all probes have fully completed growth, those `arm`s which (start-

ing from a horizontal offset relative to those probes which would be the same as if they had grown from a `top` module) can grow downward past the probes without any collisions occurring, meaning that they would never be able to place a tile, in the absence of the probes, which disagrees with a tile placed by a probe. We call these subsets of `arm`s the *probe-dodgers* of that probe set.

Definition. Given an assembly $\alpha \in \mathcal{A}[\mathcal{S}]$ containing the maximum number of tiles which can be placed in subassembly i_j (i.e. the jth subiteration of the ith iteration) without growing above the southernmost row of macrotiles of `left` or any tiles of `top`, the *signature* of a subiteration is the combination of the full specification of the macrotiles representing the fully grown `bumper`s to the left and right of the `left` module, plus the full specification of the row of macrotiles below and between them, plus the full definition of the probe-dodgers set of that subiteration.

A signature can be determined for a subiteration by growing the `planter` module to the right beyond the `right` module, the full `right` module, and as many tiles as possible between the bumpers surrounding the `left` module without growing above the first row of macrotiles in the `left` module. This can be always be done because $\mathcal{S}$ must be able to follow the dynamics of $\mathcal{T}$, in which there are assembly sequences which do exactly this.

Lemma 1. For each iteration i of an infinite set of iterations, the empty subiteration i_j of that iteration must have a signature which is identical to that of another subiteration i_k, for $j \neq k$, of that iteration.

Proof. We will prove Lemma 1 by contradiction. Therefore, assume that for no more than a constant number c of iterations do the iterations of $\mathcal{S}$ have empty subiterations which have signatures identical to some other in their iteration. Thus, for each iteration $i > c$, the empty subiteration has a signature unique among all others in its iteration.

By Lemma 3, we know that the full `left` module can grow from tiles which form on paths only from its signature, since the locations of the signature would be analogous to

γ_M of the Lemma, and this includes `bumpers` and it must be possible to grow `left` completely with or without the `bumpers`, thus any additional paths of tiles that could contribute to the growth of the `left` would have to go through the `bumpers` and thus by growing the `bumpers` the paths which would have grown through them can be continued toward the `left` module, allowing it to fully grow.

We now calculate the number of unique signatures in subiteration i_j. To fully specify the path between the macrotiles representing the bumpers to the left and right of the `left` module, we first note that that can consist of a maximum of $\log(\log(i))$ macrotiles to represent the first row of the `left` module plus another constant number of macrotiles to specify the remaining macrotiles on that path, for a total of $O(\log(\log(i)))$ macrotiles. The number of unique ways to fully specify the entire contents of $O(\log(\log(i)))$ macrotiles (which is greater than or equal to the number of ways to fully specify just the northern row), is $O(t^{m^2 \log(\log(i))}) = O(\log^c(i))$. Additionally, we note that n_{arms} is a constant independent of i (by Lemma 1), and thus that is also true for the size of the power set of all `arm` shapes, which represents the full set of possible probe-dodger sets. This means that the number of unique probe-dodger sets is constant relative to i, and therefore adding in specification of one of the constant number of probe-dodger set only allows for a constant multiplier for the number of unique signatures, resulting in a total of $O(\log^c(i))$ possible unique signatures for subiterations of iteration i.

Given the total of $O(\log^c(i))$ possibly unique signatures, we can apply Observation 1 where $|E| = O(\log^c(i))$ and $|x| = \log(\log(i))$ and note that it shows that $F(n) \in \Omega(2^n)$. However, to calculate the signatures of all subiterations of iteration i requires only that we simulate the planter in such a way as to remember only the most recent two columns at any given time, requiring a maximum space $O(i)$ (i.e. bounded by its maximum height), and also to record the signatures of the unique subiterations, requiring $O(\log^c(i)^2) < 2^i$. However, this contradicts the $F(n)$ of Observation 1, and therefore it must be the case that for an infinite number of iterations, the empty subiterations of those iterations have sig-

natures which are identical to those of at least one other subiteration in their respective iterations.

□

The following lemma states that for the empty subiteration, the arm that assembles between the macro bit-alley must have a "pinch point".

Lemma 2. For $i, j, k \in \mathbb{N}$ and some iteration i, suppose that the empty subiteration i_j and a distinct subiteration i, k have identical signatures. Additionally, let $\texttt{arm}_{i,j}$ denote the arm that assembles in the macro bit-alley of subiteration i, j, and let P_k be the left probes of $\texttt{left}^U_{i,k}$ and P'_k be the right probes of $\texttt{left}^U_{i,k}$. Then, there exists subconfigurations P_j and P'_j of $\texttt{arm}_{i,j}$ such that P_j is congruent to P_k and P'_j is a congruent subconfiguration of P'_k. Additionally, the subconfiguration C of $\texttt{arm}_{i,k}$ corresponding to $\texttt{arm}_{i,k}$ restricted to dom $(\texttt{arm}_{i,k}) \setminus (\text{dom } (P_k) \cup \text{dom } (P'_k))$ has the property that there exists a single tile t_k in C such that removing this tile from C partitions $C - t_k$ into two nonempty sets of tiles such that no two tiles of these sets are adjacent.

Proof. First, by Lemma 3, the fact that the left computation of subiteration i, j and subiteration i, k have bumpers on the left and right side, and the assumption that these subiteration have identical signature, it follows that P_j is congruent to P_k. Then, as both subiterations have the same set of probe-dodgers and this set must be non-empty since the $\texttt{arm}_{i,j}$ must assemble in the i, j subiteration, it must be the case that P'_j is a congruent subconfiguration of P'_k.

Finally, by Lemma 4, it must be the case that the gap between P_k and P'_k in i_k must be single tile wide or less in order for the probes in P_k and P'_k to cooperatively place a tile. Therefore, the subconfiguration C of $\texttt{arm}_{i,k}$ corresponding to $\texttt{arm}_{i,k}$ restricted to dom $(\texttt{arm}_{i,k}) \setminus (\text{dom } (P_k) \cup \text{dom } (P'_k))$ has the property that there exists a single tile t_k in C such that removing this tile from C partitions $C - t_k$ into two nonempty sets of tiles such that no two tiles of these sets are adjacent.

□

3.7.2 Turing machines simulating tile assembly systems

In this section we prove a couple of claims on the amount of space a Turing machine requires to simulate a system which grows certain modules of the system $\mathcal{T}$.

We call a tile an L tile if it is of type $0L$ or $1L$. We call a tile an R tile if it is of type $0R$ or $1R$. We define the bit-alley region of a subiteration i, j to be the points which lie in between $\texttt{left}_{i,j}$ and $\texttt{right}_{i,j}$ (that is, the points which lie on the same row as a tile in $\texttt{left}_{i,j}$ and $\texttt{right}_{i,j}$) and have the same x-coordinates as points which lie between the L and R tiles in subiteration i, j. We define the macro bit-alley of an iteration i, j to be the macrotile equivalent of the bit-alley region. Let (x, y) be the bottom leftmost corner of a macrotile location which lies between an L and R macrotile. We call the region $R = \{(x', y') | x - 2 \leq x' \leq x + c + 2, y - 2 \leq y' \leq y + c + 2\}$ a probing region. Let $\alpha' \in \mathcal{A}[\mathcal{T}]$. We say that a module $\gamma \sqsubseteq \alpha$ is not assembled in α' if $\text{dom}\ (\gamma) \cap \text{dom}\ (\alpha') = \varnothing$. Now, let $\alpha' \in \mathcal{A}[\mathcal{U}]$. Similarly, we say that a macro module $\gamma \subseteq \alpha_U$ is not assembled in α' if the module is not assembled in the assembly $R^*(\alpha')$.

Let $\mathcal{T}$ be TAS and let $\alpha\mathcal{A}[\mathcal{T}]$. We say that a subconfiguration $\gamma' \sqsubseteq \alpha$ grows from a subconfiguration $\gamma \sqsubseteq \alpha$ provided that there exists a path in the binding graph G_α from a tile in γ to all tiles in γ'. Here, γ and γ' are assumed to be connected. Also, we say that γ and γ' are a distance of at most 1 apart if there exist tiles $t \sqsubseteq \gamma$ and $t' \sqsubseteq \gamma'$ such that the Manhattan distance between t and t' in G_α is at most 1. Otherwise we say that the distance between γ and γ' is greater than 1.

Claim 2. Let $i, j \in \mathbb{N}$. The subconfigurations grown in the macro bit-alley from $\texttt{left}^U_{i,j}$ and $\texttt{right}^U_{i,j}$ can be output by a Turing machine M' which runs in space $2^{2|i|}$.

In this proof we rely on a straight forward adaptation of Lemma 9. The straight forward adaptation of the lemma we discuss holds because of the key insight in the proof of Lemma 9 that in order for a Turing machine to simulate a system which is simulating a zig-zag system, it only needs to "remember" a bounded number of tiles depending on the scale factor of the simulation and the width of the system that is being simulated. This

key insight allows a Turing machine to not only output the result of a computation that takes place in the simulating system, but it also allows us to construct a Turing machine which outputs the configurations contained in certain macrotile regions of the producible assemblies of the simulating system.

Proof. Note that in $\mathcal{T}$ the planter grows in a zig-zag fashion. Consequently, it follows from a straight forward adaptation of Lemma 9 that the configuration of the row of macrotiles which compose the first rows of the $\texttt{left}^U_{i,j}$ and $\texttt{right}^U_{i,j}$ can be determined in space $O(i^2)$ since the width of the counter in $\mathcal{T}$ is $O(i^2)$.

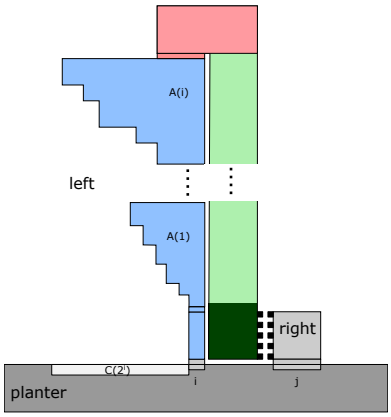


Figure 3.4: A depiction of a portion of the $\texttt{planter}$, $\texttt{left}_{i,j}$, and $\texttt{right}_{i,j}$ of the configurations of the (i,j)-subiteration in $\mathcal{T}$. The blue, red, light green, and dark green regions of $\texttt{left}_{i,j}$ correspond to various subassemblies of $\texttt{left}_{i,j}$. The dark green region is an $i \times i$ block of tiles which can be determined in $O(2^i)$ space.

Figure 3.4 shows a portion of the $\texttt{planter}$, $\texttt{left}_{i,j}$, and $\texttt{right}_{i,j}$. The configuration $\texttt{left}_{i,j}$ has been divided into five regions.

1. The blue region corresponds to the zig-zag assembly which computes $A^+(i)$,
2. the red region corresponds to the assembly which turns the i bits calculated by $A^+(i)$ toward the counter,
3. the light green region corresponds to the zig-zag counter assembly which grows toward the $\texttt{planter}$, counting to 2^{2^i} before growing the probes of the left side of the bit-alley, and

4. the dark green region which corresponds to the subassembly containing the left portion of the bit-alley.

Now consider the subconfiguration L in $\mathcal{U}$ which represents the subassembly corresponding to the dark green region in Figure 3.4. A straightforward adaptation of the proof of Lemma 9 shows that L can be determined in $O(2^i)$ space. This adaptation consists of modifying the proof so that the glue sequence tables and the assemblies produced by the procedures InitAssembly, InitGST, UpdateAssembly, and UpdateGST follow the zig-zag assembly sequence of the subassembly in the blue region before turning and following the zig-zag assembly sequence of the light green region. Both of these zig-zag assemblies have width in $O(2^i)$. Consequently, there exists a Turing machine which runs in space $O(2^i)$ and outputs the configuration in the dark green region of $\texttt{left}^U_{i,j}$. Then, as $\texttt{right}^U_{i,j}$ consists of $O(i^2)$ tiles, the claim holds. □

Claim 3. Let $i, j \in \mathbb{N}$ be such that i, j is an empty subiteration in α_U. Let $\alpha'_U \in \mathcal{A}[\mathcal{U}]$ be such that the $\texttt{left}^U_{i,j}$ and $\texttt{right}^U_{i,j}$ of subiteration i, j is assembled, but the $\texttt{top}^U_{i,j}$ module has not been assembled. Then if α'_U contains configurations P and P' which grow from $\texttt{left}^U_{i,j}$ and $\texttt{right}^U_{i,j}$ respectively in the macro bit-alley of subiteration i, j which are at most a distance of 1 apart, then there exists a Turing machine M' which takes α'_U as input and outputs a set of t arm types, denoted A', such that the arm which grows in subiteration i, j in α_U is in the set A'. Furthermore, M' runs in space $O(|U| \times c^2)$.

Proof. Let the hypotheses hold, and assume that γ is the subconfiguration grown in the probing region R which contains P and P'. By assumption there exists subconfigurations P and P' which are a distance of 1 apart. We can use these two subconfigurations to construct $|U|$ different systems $\mathcal{T}_t = (U, \sigma_t, \tau)$. For each $t \in U$ the system $\mathcal{T}_t$ is constructed by constructing σ_t so that it consists of γ with the tile type t placed in the single tile wide gap between P and P'. Note that there could be more than one single tile wide gap between P and P'. It doesn't matter which one we choose as long as we choose the same one when constructing different systems. We can then simulate the growth of all the systems with

a Turing machine M' in the following manner. For each $\mathcal{T}_t$, M' simulates the assembly of the system until two full macrotile regions form. At that point, the Turing machine is then able to determine what tile types the macrotiles map to in T. Observe that since the configuration $\gamma \cup t$ "cuts" the bit-alley region, and an arm must be able to grow in the subiteration since it is assumed to be empty, there exists at least one t such that $\gamma \cup t$ which grows into two full macrotile regions and does not place any tiles outside of the bit-alley. From this, the Turing machine can determine what types of arms are able to grow into the subiteration. This Turing machine can clearly be designed to run space $O(|U| \times c^2)$. □

3.7.3 A contradiction

Let the B^+ machine be defined analogously to the A^+ machine described in Section 3.6, but for the series of computations of B. Note that there does not exist a machine which outputs $B^+(x)$ and runs in time $G(n)$ for $G(n) \in o(3^{|2n|})$ for infinitely many inputs $x \in \mathbb{N}$. This follows from the description of the languages that the machines B^+ decides which are described in Section 3.6.

In this section, we show that under the hypothesis there exists a simulator $\mathcal{U}$ for $\mathcal{T}$ we can construct a Turing machine M^* which outputs $B^+(x)$ and runs in time $G(n)$ for $G(n) \in o(3^{2n})$ for infinitely many inputs $x \in \mathbb{N}$. This will contradict the assumption that the language L_B cannot be recognized in i.o. space complexity $G(n)$ for $G(n) \in o(3^{2n})$.

For each input $x \in \mathbb{N}$, M^* does the following. First, M^* simulates the growth of an assembly $\alpha_e \in \mathcal{A}[\mathcal{U}]$ such that α_e only grows the `planter` and signatures (which means that α_e also contains `right` modules) in order to determine whether the empty subiteration in iteration x is unique. Note that this requires space $O(|x|)$ by Lemma 8 since this is the space used by the `planter` in $\mathcal{T}$. If the empty subiteration in iteration x is unique, the machine M^* simply runs the machine B^+ on input x, outputs $B^+(x)$ and halts. That is, compared to $B^+(x)$, M^* with input x is no more space efficient in this case.

Creating a set E of t arm types

We refer to the tile types of T that assemble the various `arm`s in $\mathcal{T}$ as arm types. By an abuse of notation, we refer to the macrotile that maps to an arm type as an arm type of $\mathcal{U}$. Then, if the empty subiteration is non-unique M^* creates a set E of t arm types such that the arm type which grows into the empty subiteration i_j is guaranteed to be in E. It does so in the following manner. Denote the empty subiteration by x_j and denote the non-empty subiteration with an identical signature by x_k. Next, the TM M^* determines the left and right probes grown by $\texttt{left}^U_{x_k}$ and $\texttt{right}^U_{x_k}$. It follows from Claim 2 that this can be done in space $2^{2|x|}$. It follows from Lemma 2 that any left probe P of $\texttt{left}^U_{x,k}$ and any right probe P' of $\texttt{right}^U_{x,k}$ are such that the $\texttt{arm}^U_{i,j}$ is *consistent* with the translation of P and P' by some $\vec{v}$. That is, they do not have different tiles in the same location after the translation.

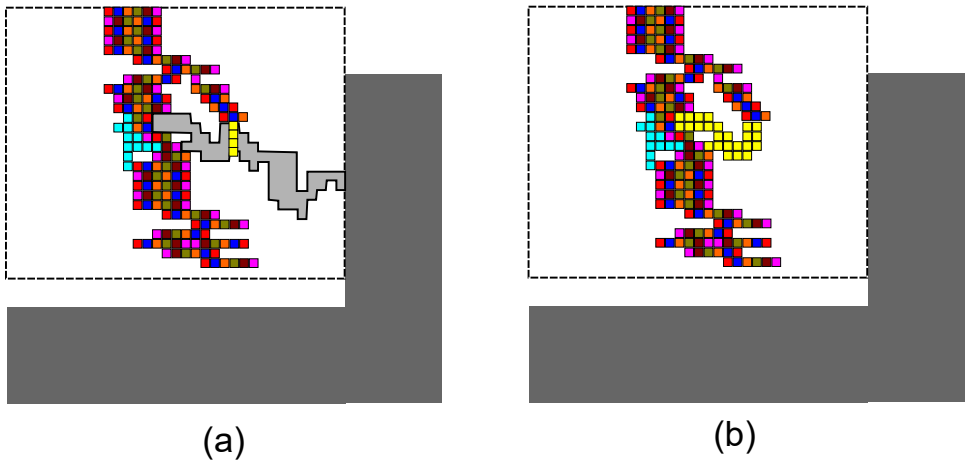


Figure 3.5: If a subconfiguration γ (shown as the yellow strip in part (a)) grows a strip of tiles which is consistent with and completely spans another subconfiguration γ', then whatever grows in γ' after the strip can grow as shown in part (b).

Note that at this point we know that there is some strip of $\texttt{arm}^U_{x,j}$ that contains tiles in P and is completely consistent with the subconfiguration P'. Note that it follows from Lemma 3 that if $\texttt{arm}^U_{x,j}$ grows a subconfiguration of tiles which completely cuts the region where it must be consistent with P' as shown in Figure 3.5 part (a), then it must be

the case that it can grow all of P' which grows after the cut as shown in part (b) of Figure 3.5. Furthermore, note that this growth can occur without any tile placements outside of dom (P'). Consequently this means if we want to create the set E, all we need to do is for each tile $u \in U$, create a system which contains only the probe P and the tile u placed at position p_t. We then grow the system until the assembly which we obtain is terminal or the diameter of the assembly we obtain is greater than $2c$. If the diameter of the assembly is greater than $2c$ and some portion of it maps to an arm tile t_a under the representation function, then we add t_a to our set E. Let A be a subconfiguration and let R be the infinite which consists of infinite columns such that dom $(A) \subset R$ and if any column is removed from R, it is no longer true that dom $(A) \subset R$. Then we call the southern boundary of A the set of points $x \in$ dom (A) such that x contains a path p completely contained in R to a point which lies to the south of any point in dom (A). Intuitively, this is the set of tiles on the "bottom path" of A. Note this does not just include tiles in A which have an empty location to the south. If the assembly is terminal and has diameter less than $2c$, we then create a new seed which consists of the previous seed and the tile t_s in P' such that 1) it is not contained in the previous seed, 2) it is one of tiles in the southernmost boundary of P') and 3) the path of points contained in the southern boundary tiles from t_s to p_t, denoted ∂_s, is such that all the points in ∂_s were in the previous seed. We then repeat this process for all $u \in U$.

The Turing machines M^* runs the algorithm shown in Algorithm 1. This algorithm is just a formalization of the intuitive idea discussed above. Here are the variables we use in this algorithm:

1. P is the left probe grown from $\texttt{left}^U_{x,k}$.
2. P' is the right probe grown from $\texttt{right}^U_{x,k}$.
3. p_t is a point which lies adjacent to both P and P' (i.e. the single tile wide gap between P and P').

4. $P^* = \text{dom}\,(P) \cup \text{dom}\,(P') \cup p_t$.

5. ∂_{P^*} is the *southern boundary* of P^*.

6. Intuitively Q is the min queue which contains tiles on the southern boundary of P' and they are added to the queue based off of how far they are away from p_t.

Input: P, P', U, p_t as described above
Output: A set E of arm types of size at most $|U|$.
set Q to be a min queue of tiles in $\partial_{P'}$ where the key for a tile t' is the length of the path from dom (t') to $p_{r'}$ in the grid graph restricted to only points in ∂_{P^*};
for $t^* \in U$ **do**
  set $\mathcal{T}^*$ to the system (U, σ_t, τ) where σ_t is the assembly P with tile t^* placed at point p_t;
  do
    grow $\mathcal{T}$ until the assembly α is terminal or $\text{diam}(\alpha) > 2c$;
    if $\text{diam}(\alpha) > 2c$ **then**
      break;
    end
    else
      $\sigma_t = \sigma_t \cup \text{pop}(Q)$ (that is, redefine σ_t to be the assembly P with tile t^* placed at point p_t and the tile popped from Q at its tile location;
    end
  while Q *is not empty*;
  if α *contains a macrotile which maps to an arm tile,* t_a *say* **then**
    add t_a to E;
  end
end

Algorithm 1: An algorithm for constructing the arm types for a non-unique empty subiteration.

Using E to compute $B^+(x)$

Note that the arm tiles in E correspond to the output of the machines B_i, so we can think of each arm tile in E as corresponding to a string $x_0x_1x_2...x_{2t-1}$ where x_i represents the output of M_{B_i} on the input received by `top`. Once $M*$ determines the t potential arms that can grow into the empty subiteration x, j, it can create a set of t strings of length $2t$ which correspond to the arms. Furthermore, it must be the case that one of these t strings corresponds to the output of B^+ on input x. Thus, we have a set of t strings of length $2t$

which contains the solution to $B_0(x)B_1(s(x,1))\dots B(s(x,|x|))$. Then by Lemma 5, for almost all inputs, M^* requires at least space 3^n, but as we observed M^* only uses space 2^{2n}). This is a contradiction.

Proof of correctness for the algorithm which generates E

We begin by noting that the set E produced by Algorithm 1 is guaranteed to contain a translation of a macrotile grown in $\texttt{arm}^U_{x_j}$. The algorithm implicitly described above is guaranteed to grow a macrotile that the arm grows for the following reasons. Let t_p be the tile such that the arm $\texttt{arm}^U_{x_j}$ is consistent with the configuration $P \cup t_p \cup P'$ and let ∂_{P^*} be the southern boundary of this configuration. We show that there is an assembly sequence in $\mathcal{U}$ such that the tiles placed in ∂_{P^*} before growth continues to the south of ∂_{P^*} is the same as some seed in our algorithm. Let P_j and P'_j be the subconfigurations grown from $\texttt{left}^U_{x,j}$ and $\texttt{right}^U_{x,j}$ respectively such that $P_j \cup P'_j \sqsubseteq\!\!\!\approx\ P \cup P'$. First, there is guaranteed to be an assembly sequence where P_j is present since the assumption that subiteration i_j and subiteration i_k have the same signatures implies that there exists an assembly where their $\texttt{left}^U$ modules are exactly the same up to translation. This means that there exists an assembly where P_j is present before $\texttt{arm}^U_{x_j}$ grows into the $\texttt{bitAlley}^U_{x_j}$. In addition, there is an assembly sequence where first P_j appears (because of the previous point), and then a translation of t_p appears next in ∂_P so that it prevents the cooperative growth of the macrotile which grew from the probes in the subiteration x_k (otherwise the macrotile grown in the simulated bit-alley of iteration x_k could assemble). Finally, there exists an assembly sequence where after $\texttt{arm}^U_{x_j}$ places a macrotile on the southern boundary of dom (P'_j) all of the tiles in P'_j which lie to the west of the strip can grow with only tile placements in dom (P'_j).

Now, notice that when U and c are fixed, Algorithm 1 runs in constant time.

Finally, we claim that the algorithm adds at most t arm types to the set E. Indeed, this is true since it is clear from the algorithm that at most one arm type can be added to E with each iteration of the outer loop of which there are $|U|$.

3.8 Technical lemmas

In this section we prove a number of technical lemmas which will be of assistance in later sections. The first technical lemma we prove shows that in a directed system if a subconfiguration γ "grows through" another subconfiguration γ', then the subconfiguration γ' can grow the portion of γ that assembles after γ "grows through" γ'. The second technical lemma roughly states that the growth of macrotile that represents a bit-alley tile in $\mathcal{T}$ must stem from the cooperative placement of tiles by subconfigurations grown from the left and right machines. In the third lemma, we show that in a system $\mathcal{U} = (U, \sigma, \tau)$ the number of assemblies that can grow from subconfigurations which are exactly the same for all but one tile is no more than $|U|$. This lemma will be used in a later section to show that the number of arms that can grow into an empty subiteration which has at most a one tile wide gap is at most $|U|$. The next technical lemma we prove in this section shows that if a Turing machine M is able to narrow down the solution space for the outputs of a series of computations on some input x, then M must use the same amount of space as some of the computations in the series. This lemma will allows us to put constraints on the types of tricks the adversary is able to use in order to simulate the system $\mathcal{T}$. Finally, we will prove a technical lemma which proves that the space complexity of computations possible within an assembly simulating a zig-zag assembly is asymptotically no greater than the space complexity of the system being simulated.

3.8.1 Miscellaneous definitions

Definition. Let $R \subset \mathbb{Z}^2$ be an $m \times n$ rectangular region with the bottom leftmost corner at location $(0, 0)$. Then

1) $perim_S = \{x | x = (p, 0), p \in [0, n]\}$,

2) $perim_E = \{x | x = (n, p), p \in [0, m]\}$,

3) $perim_N = \{x | x = (p, m), p \in [0, n]\}$, and

4) $perim_W = \{x | x = (0, p), p \in [0, m]\}$.

Definition. Let $\mathcal{T} = (T, \sigma, 2)$ be a directed TAS and let $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$. Let $\gamma \sqsubseteq \alpha$ and $\beta \sqsubseteq \alpha$. Then we say γ and β are congruent and write $\gamma \cong \beta$ if $|\text{dom } (\beta)| = |\text{dom } (\gamma)|$ and there exists $\vec{v} \in \mathbb{Z}^2$ such that for all $x \in \text{dom } (\gamma), \gamma(x) = \beta(x + \vec{v})$.

Definition. Let $\mathcal{T} = (T, \sigma, 2)$ be a directed TAS and let $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$. Let $\gamma \sqsubseteq \alpha$ and $\beta \sqsubseteq \alpha$. Then we say γ is a congruent subconfiguration of β and write $\gamma \sqsubseteq_{\cong} \beta$ if there exists a subconfiguration $\beta' \sqsubseteq \beta$ such that $\gamma \cong \beta'$.

Let $i, j \in \mathbb{N}$ We call P a left (right) probe if P is grown from $\texttt{left}_{i_j}^S$ ($\texttt{right}_{i_j}^S$) and there exists $P' \sqsubseteq \alpha^S$ such that P' grows from $\texttt{right}_{i_j}^S$ ($\texttt{left}_{i_j}^S$) and P and P' are a distance of 1 apart. Throughout this paper, we assume α^S is the single terminal assembly of $\mathcal{S}$.

3.8.2 Path-crossing subconfigurations

We now show that in a directed system if a subconfiguration γ "grows through" another subconfiguration γ', then the subconfiguration γ' can grow the portion of γ that assembles after γ "grows through" γ'.

We now begin with some definitions to allow us to more concretely define what it means for a subconfiguration to grow another subconfiguration and what it means for a subconfiguration to grow through another subconfiguration. As we will see, the intuitive notion of a subconfiguration γ growing a subconfiguration γ' means that there exists a path in the directed binding graph from γ to γ'. Also, we will see that the idea of a subconfiguration γ growing through a subconfiguration γ' means that all paths in the binding graph from γ "cut" through the subconfiguration γ'.

Definition. Let $\mathcal{T}$ be a directed TAS such that $\vec{\alpha}$ is an assembly sequence of $\mathcal{T}$. We now define the directed binding graph of an assembly sequence $\vec{\alpha}$ which we denote by $G_{\vec{\alpha}}$. The vertices of the directed binding graph $G_{\vec{\alpha}}$ are tiles in $res(\vec{\alpha})$ and there is an edge from tile t to tile t' in $G_{\vec{\alpha}}$ provided that a glue on t serves as an input glue to t'.

Definition. Let $\mathcal{T}$ be a directed TAS such that $\vec{\alpha}$ is an assembly sequence of $\mathcal{T}$. Also let $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$ and let $\gamma \sqsubseteq \alpha$. Let p be a path from a tile t to a tile t' in $G_{\vec{\alpha}}$ such that p

contains tiles that belong to a connected subconfiguration γ. Then we say that the path p cuts γ provided that the subgraph of $G_{\vec{\alpha}}$ which contains only tiles in γ is disconnected when the vertices in p are removed.

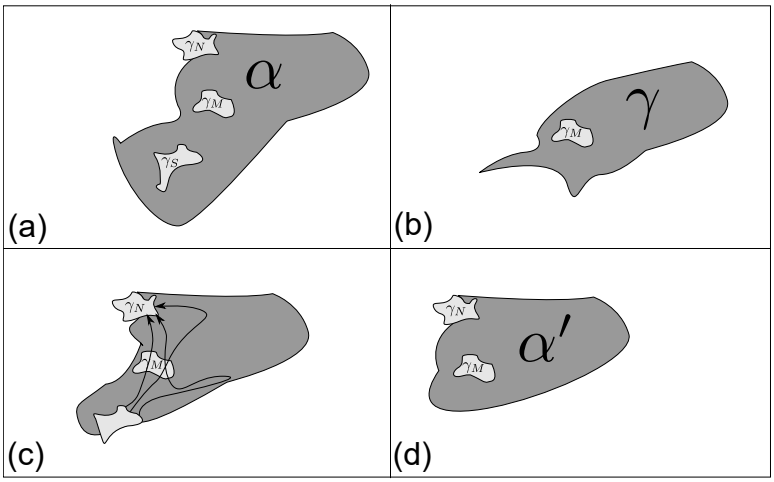


Figure 3.6: A schematic diagram of the assemblies and subconfigurations discussed in Lemma 3. Part (a) shows the subconfigurations γ_S, γ_M, γ_N and the terminal assembly α. Part (b) shows a schematic representation of the assembly γ from condition (1) of the lemma statement. Part (c) shows a schematic representation of all the paths in the binding graph of $G_{\vec{\alpha}}$ from tiles in γ_S to γ_N. Part (d) shows the producible assembly α' in the conclusion of the statement.

Figure 3.6 shows the schematic representation of the conditions listed in the statement of Lemma 3 and it's conclusion. Intuitively, the first condition of Lemma 3 states that the growth of γ_M is not dependent on the growth of γ_S. Consequently, in our schematic representation, this means that there exists an assembly which looks like the one shown in part (b) of Figure 3.6. The second condition of the lemma statement says that there exists some assembly sequence such that γ_N grows independently of γ_S or γ_S always grows through γ_M to grow γ_N. This is represented schematically in part (c) of Figure 3.6. The result of Lemma 3 is that γ_N can be grown without growing γ_S which is represented schematically in part (d) of Figure 3.6.

Lemma 3. Let $\mathcal{T}$ be a directed TAS and let $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$. If $\gamma_N, \gamma_M, \gamma_S \sqsubseteq \alpha$ are subconfigurations such that

1. there exists $\gamma \in \mathcal{A}[\mathcal{T}]$ such that $\gamma_M \sqsubseteq \gamma$ and for all $x \in \text{dom } (\gamma_S)$, $x \notin \text{dom } (\gamma)$, and
2. there exists $\vec{\alpha}$ of $\mathcal{T}$ such that $\text{res}(\vec{\alpha}) = \alpha^*$ where $\gamma_N, \gamma_M \sqsubseteq \alpha^*$ and for all paths p in $G_{\vec{\alpha}}$ from a tile in γ_S to a tile in γ_N, p cuts γ_M,

then there exists $\alpha' \in \mathcal{A}[\mathcal{T}]$ such that $\gamma_M, \gamma_N \sqsubseteq \alpha'$ and for all $x \in \text{dom } (\gamma_S)$, $x \notin \text{dom } (\alpha')$.

Proof. Let the hypotheses hold. We now construct an assembly sequence $\vec{\beta}$ which contains an assembly α' such that $\gamma_M, \gamma_N \sqsubseteq \alpha'$ and for all $x \in \text{dom } (\gamma_S)$, $x \notin \text{dom } (\alpha')$. Let the assembly sequence $\vec{\gamma}_{i=0}^{i=k}$ be an assembly sequence in $\mathcal{T}$ such that $res(\vec{\gamma}) = \gamma$. We construct $\vec{\beta}$ by passing $\vec{\alpha}$ and $\vec{\gamma}$ as arguments to Algorithm 2 and store the output of the algorithm in $\vec{\beta}$. Note that for all $\beta_i \in \vec{\beta}$, $\beta_i \longrightarrow^t \beta_{i+1}$ is valid since 1) the algorithm ensures that

Input: $\vec{\gamma} = (\gamma_0, \gamma_1, ..., \gamma_k)$, $\vec{\alpha} = (\alpha_0, \alpha_1, ...)$
Output: $\vec{\beta} = (\beta_0, \beta_1, ...)$
for $i \in [0, k]$ **do**
  $\beta_i := \gamma_i$;
end
for $i \in [1, |\vec{\alpha}|]$ **do**
  $t := \text{dom } (\alpha_i) \setminus \text{dom } (\alpha_{i-1})$;
  if $t \notin \text{dom } (res(\beta))$ *and* $t \notin \text{dom } (\gamma_S)$ **then**
    $\beta_{k+i} := \beta_{k+i-1} + t$;
  end
  unless there exists a path from γ_S to t in $G_{\vec{\alpha}}$ which does not cut γ_M;
end

Algorithm 2: An algorithm for constructing $\vec{\beta}$.

$t \notin \beta_i$ and 2) t attaches with strength τ since the algorithm ensures all of $t's$ input glues are present. Also note that since it is assumed that for any tile t in γ_N all paths from γ_S to t cut γ_M, there exists an assembly $\alpha'' \in \vec{\beta}$ such that $\gamma_M \sqsubseteq \alpha''$, $\gamma_N \sqsubseteq \alpha''$ and for all $x \in \text{dom } \gamma_S$, $x \notin \text{dom } (\alpha'')$. □

3.8.3 Necessity of probes

We say a macrotile is an L-macrotile or an R-macrotile if, under the representation function R, the macrotile maps to either a 0_L or 1_L tile type or either a 0_R or 1_R tile type respectively. Additionally, we say a macrotile is a bit-alley macrotile if the macrotile maps

to either a 1_M, 1_B, 0_M, or 0_B tile under the representation function. Finally, we call the $m \times (m+4)$ region which consists of the macrotile region between L-macrotiles and R-macrotiles extended by two tile widths to the west and east the *probing region*.

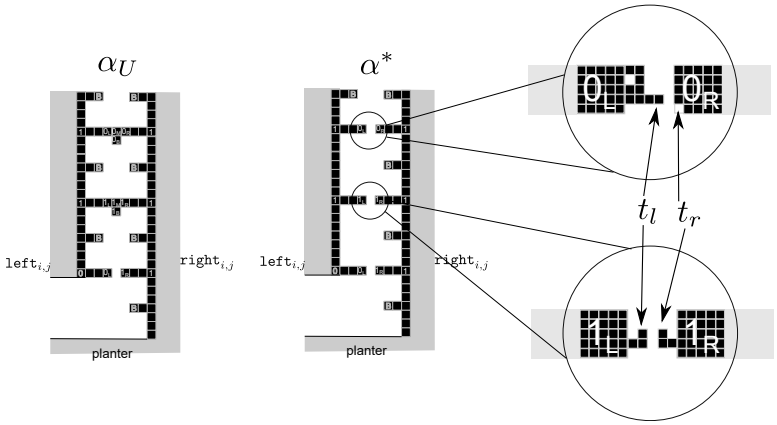


Figure 3.7: An example of subconfigurations of assemblies discussed in Lemma 4 along with the tiles t_l and t_r also discussed in the lemma. This left part of this figure shows the `bitAlley` subconfiguration of subiteration i, j in $\alpha^S \in \mathcal{A}_{\square}[\mathcal{S}]$. The right part of this image shows the `bitAlley` subconfiguration of subiteration i, j in $\alpha^* \in \mathcal{A}[\mathcal{S}]$ and also shows a zoomed in view which shows examples of the tiles t_l and t_r as described in the lemma statement.

Next we show a lemma which intuitively says that for a non-empty subiteration, subiteration i, j say, a valid simulation of $\mathcal{T}$, it must be the case that simulated probes on the right side of $\texttt{left}^S_{i,j}$ and the simulated probes on the left side of $\texttt{right}^S_{i,j}$ must in fact cooperate. Referring to Figure 3.7, in order for $\mathcal{S}$ to simulated $\mathcal{T}$, the macrotiles 0_L and 0_R depicted in the figure must assemble probes that come within one tile of each other to allow for cooperation across the simulated bit-alley. The high-level idea is that the macrotile 0_M cannot assemble to represent a non-empty tile of T until the macrotiles 0_L and 0_R have assembled. Therefore, the macrotiles 0_L and 0_R must "coordinate". Moreover, this coordination cannot be the result of growing a path through the macrotile regions corresponding to simulations of bumper tiles.

Lemma 4. Let $i, j \in \mathbb{N}$. There exists $\alpha^* \in \mathcal{A}[\mathcal{S}]$ such that 1) $\texttt{left}_{i,j}$ and $\texttt{right}_{i,j}$ modules are grown without assembling any bit-alley macrotiles, and 2) if α^S, the terminal assem-

bly of $\mathcal{S}$, contains a bit-alley macrotile in the probing region R, then α^* contains the tiles t_l and t_r in the probing region R which grow from an L macrotile and R macrotile respectively such that t_l and t_r are at most one tile width apart.

Proof. Let C denote the (i, j) subconfiguration. Let $\beta_{i,j}$ be an assembly in $\mathcal{A}[\mathcal{T}]$ such that C contained in $\beta_{i,j}$ has assembled to where every tile of type 1_M, 1_B, 0_M, or 0_B that can bind has. Now, suppose that no tile of type B in this subconfiguration has attached. Refer to Figure 3.3 for a depiction of these tiles types and the tile locations where they bind in a subconfiguration. Under the assumption that $\mathcal{S}$ simulates $\mathcal{T}$, it must that there exists β' in $\mathcal{A}[\mathcal{S}]$ such that $R^*(\beta') = \beta$. We now use β' to construct α^*. First, we have the following observation.

Claim 4. For each probe region R of $\beta'_{i,j}$ that contains a bit-alley macrotile, there must be a path of adjacent tiles with matching glues along their adjacent edge which starts with a tile at a westernmost location in R and ends with a tile at an easternmost location of R.

We prove Claim 4 by contradiction. Therefore, suppose that there exists a probe region R in $\beta'_{i,j}$ that contains a bit-alley macrotile such that there is no path of adjacent tiles with matching glues along their adjacent edge which starts with a tile on the west edge of R and ends with a tile on the east edge of R. Let μ and η be the macrotiles subassemblies of $\beta'_{i,j}$ that map to 1_M or 0_M and 1_B or 0_B respectively. Note that no tile of η can bind until the macrotile region M above η maps to 1_M or 0_M under R, for otherwise $\mathcal{S}$ is not a valid simulation. Moreover, M must map to the empty tile until the macrotile to west of M maps to either a 0_L or 1_L tile type and the macrotile to the east of M maps to either a 0_R or 1_R tile type.

Under the assumption that there is no path of adjacent tiles with matching glues along their adjacent edge which starts with a tile at a westernmost location of R and ends with a tile at an easternmost location of R, it must be the case that there is a cut v of R such that the sum of all glue strengths corresponding to glues of adjacent tiles on each side of this cut is zero.

Therefore, in any assembly sequence of $\beta'_{i,j}$ there must be a path p of adjacent tiles starting from a tile in the L-macrotile and ending with a tile in the R-macrotile. p cannot cross the cut v since the sum of all glues along this cut is zero. Hence, p must contain tiles in macrotile regions that map to B tile types or map to tile types of the `left` or `right` module that adjacent to B tile types. In any case, now consider the assembly sequence in $\mathcal{T}$ where tiles of type B always bind before tiles of type 0_R, 1_R, 0_L, or 1_L. In the simulation of this sequence, it must be the case that the macrotile regions mapping to B tile types or to a tile adjacent to a B tile type contain enough tiles of p to assemble the portion of p starting from the macrotile regions mapping to B tile types or to a tile adjacent to a B tile type and ending with a tile of η. This follows from Lemma 3. This portion of the path p either contains a tile in an L-macrotile or an R-macrotile. Suppose this portion of the path p contains a tile in an L-macrotile (the R-macrotile case is similar). Then, note that this portion of the path can assemble even in the absence of any tiles of the R-macrotile. Therefore, there is an assembly sequence of $\mathcal{S}$ such that a tile of η binds before any tiles of the R-macrotile bind, which is before M can map to 1_M or 0_M. This violates valid simulation of $\mathcal{T}$ by $\mathcal{S}$ since the tile of η that binds is outside of any fuzz region. Therefore, Claim 4 holds.

To finish the proof of Lemma 4, start with the assembly β'. By Claim 4, it must be the case that for each probing region R, there is no strength zero cut of R separating tiles of $\beta'|_R$. Hence, we can obtain α^* as follows. For each probing region R, if a single tile wide gap does not exist in a probing region R, remove a tile from the subassembly of β' contained in R in such a way that the resulting assembly is still valid until there is a single tile wide gap. This tile removal corresponds to "rewinding" the assembly sequence of β' in the region R just to the point were there is a single tile wide gap between tiles stably attached to tiles of an L-macrotile and tiles stably attached to tiles of an R-macrotile. □

3.8.4 Narrowing down the outputs of a set of Turing machines

Let R be the Turing machine defined in Section 3.6.1. The next lemma essentially says that if there is a Turing machine M' which outputs a set of strings E such that the of TM R on input x is in E, then the TM M' must use at least space $3^{|x|}$. A similar statement also holds for the Turing machine A^+ defined in Section 3.6.1.

Lemma 5 is used in Section 3.7.3. In Section 3.7.3, we must show that in some empty subiteration i, j, the modules $\texttt{left}^S_{i,j}$ and $\texttt{right}^S_{i,j}$ cannot somehow narrow down the arms which are going to be grown from the $\texttt{top}_{i,j}$ and grow the probes in the `bitAlley` region so that the probes which they grow are "consistent" with the arm which will grow from $\texttt{top}_{i,j}$. In Section 3.7.3 we show that it is impossible for the adversary to narrow down the arms which will grow from $\texttt{top}_{i,j}$ to a set of t arm types. We use Lemma 5 to show this. Intuitively, this lemma says it is impossible for the adversary to narrow down the arms which can grow for the following reasons. First note, that there are 2^t possible arm types that can grow from $\texttt{top}_{i,j}$ each of which represents the output of the series of B computations on input i. So, we can think of this set of arm types as a set of strings which correspond to the output of the series of B computations on input i, and we denote this set by E. Also, recall that the series of B computations require space $O(3^{2|i|})$ and the $\texttt{left}^S_{i,j}$ module has space $O(2^{2|i|})$. Suppose the adversary can narrow down the arm type grown from the $\texttt{top}^S_{i,j}$ module to t choices (implying $|E| = t$). If there exists a bit position k such that all the strings in set E agree on bit position k, then the adversary knows the output of machine B_k on input i, which contradicts the space complexity of the language decided by B_k. So it must be the case that for bit position k in every $X^0 \in E$ there exists a string $X^1 \in E$ so that X^0 and X^1 disagree on bit position k. But if this is the case, note that the adversary could then run at most the first $|E|$ B computations to prune his set E down to a single string which must contain the answer to the last computation of B which requires asymptotically more space. This implies the adversary is able to recognize the language decided by B in less space than required which is impossible. So, it must be the case he can't

narrow down the set of arms that can grow from $\texttt{top}_{i,j}^S$ to t choices.

The preceding discussion highlighted the main idea of Lemma 5 which we sate and prove now.

Lemma 5. Let R, B, s, and a be as defined in 3.6.1, and let B' be a TM which on input x outputs a set of $m < a$ strings $E \subset \{0,1\}^a$ such that there exists $X \in E$, so that $R(x) = X$. Then for an infinite number of $x \in \{0,1\}^*$, R' requires space at least $3^{|x|}$.

Proof. Assume the hypothesis. We now show that we can construct a machine M which is composed of the computation R' along with another computation M'. We show that M is able to compute the i^{th} bit of output of R. Furthermore, we show that M' uses at most space $3^{|x|+i-2}$ to compute $B(s(x,i))$ for an infinite number of x. Since computing this requires more than space $3^{|x|+i-2}$ for almost all x, this implies that R' requires space at least $\frac{1}{2}3^{|x|+i-1}$.

We now describe the machine M. On input x, M begins by removing any duplicate strings in E and then it simulates the machine R' to generate a set of $m < a$ strings. Next, M simulates the machine M' (described below) which takes the m strings output by R' as input and outputs a bit of the computation $R(x)$.

Input: A string x and a set $E \subset \{0,1\}^a$ s.t. $|E| < a$ and $R(x) \in E$
Output: A tuple consisting of an integer $0 \leq i < a$ and the i^{th} bit of $R(x)$
for *i in* $\{1, 2, ..., a-1\}$ **do**
  if *the* i^{th} *bit of all strings in E are the same* **then**
    output i, the value of the i^{th} bit and halt;
  else
    Compute $B(s(x, i-1))$ and store its output in b ;
    Remove any string in E whose i^{th} bit is not equal to b ;
    Clear the work tape;
  end
end

Algorithm 3: The algorithm performed by M'

The machine M' implements the algorithm shown in 3. We now prove that the machine M' will always output a tuple consisting of an integer $0 \leq i < a$ and the i^{th} bit of $R(x)$. To see this, note that each iteration of the for loop does one of two things. It either halts

and outputs the tuple, or it removes at least one string from the list of penitential outputs of R (i.e. the set E). By our assumption that $R(x)$ is in the set of strings output by $R'(x)$ we are guaranteed that at least one string will not be removed from the set E. And, our assumption that R' outputs less than a strings (and recall a is the length of the strings) ensures that at some point during the computation, the i^{th} bit of all strings left in E will agree. Indeed, with each iteration of the for loop, if some strings disagree on the i^{th} bit position, at least one of them will be removed. Since, this can occur $a-2$ times and there are at most $a-1$ strings in the set, we are guaranteed that at some point all the strings will agree on the i^{th} bit position.

Note that the machine M' is able to compute the i^{th} bit of $R(x)$ using space $3^{|x|+i-2}$. Indeed, for each $0 \leq k < i$, M' simulates $R(s(x,k))$ which requires at most space $3^{|x|+k-2}$, and between each simulation the work tape is cleared for reuse.

To see that M' requires space at least $3^{|x|}$ for an infinite number of x, note that we can use the machine M to compute $B(x)$ for an infinite number of x. If M' used less space than $3^{|x|}$, we would be able to calculate the output of B for infinitely many x using less space than the space hierarchy theorem requires. □

The idea behind Observation 1 is the same as that behind Lemma 5, but is more generalized to allow the series of Turing machines and set of "guesses" to change size with the input. This will be needed in the proof of Lemma 1.

Observation 1. Let A^+, A, and S be as defined in 3.6.1, and let A' be a TM which on input x outputs a set of $m < 2^{|x|}$ strings $E \subset \{0,1\}^{2^{|x|}}$ such that there exists $X \in E$, so that $A^+(x) = X$. Then for an infinite number of $x \in \{0,1\}^*$, A' requires space at least $2^{|x|}$.

The proof of 1 is identical to that of 5 with A^+ substituted for R, A substituted for B, and the space bounds changed. We present the proof below for completeness.

Proof. Assume the hypothesis. We now show that we can construct a machine M which is composed of the computation A' along with another computation M'. We show that M

is able to compute the i^{th} bit of the output of A^+. Furthermore, we show that M' uses at most space $2^{|x|+i-2}$ to compute $A(s(x,i))$ for an infinite number of x. Since computing this requires more than space $2^{|x|+i-2}$ for almost all x, this implies that A' requires space at least $\frac{1}{2}2^{|x|+i-1}$.

We now describe the machine M. On input x, M begins by removing any duplicate strings in E and then it simulates the machine A' to generate a set of $m < 2^{|x|}$ strings. Next, M simulates the machine M' (described below) which takes the m strings output by A^+ as input and outputs a bit of the computation $A^+(x)$.

Input: A string x and a set $E \subset \{0,1\}^{2^{|x|}}$ s.t. $|E| < 2^{|x|}$ and $A^+(x) \in E$
Output: A tuple consisting of an integer $0 \leq i < 2^{|x|}$ and the i^{th} bit of $A^+(x)$
for *i in* $\{1, 2, ..., 2^{|x|} - 1\}$ **do**
  if *the* i^{th} *bit of all strings in E are the same* **then**
    output i, the value of the i^{th} bit and halt;
  else
    Compute $A(s(x, i-1))$ and store its output in b ;
    Remove any string in E whose i^{th} bit is not equal to b ;
    Clear the work tape;
  end
end

Algorithm 4: The algorithm performed by M'

The machine M' implements the algorithm shown in 4. We now prove that the machine M' will always output a tuple consisting of an integer $0 \leq i < 2^{|x|}$ and the i^{th} bit of $A^+(x)$. To see this, note that each iteration of the for loop does one of two things. It either halts and outputs the tuple, or it removes at least one string from the list of penitential outputs of A^+ (i.e. the set E). By our assumption that $A^+(x)$ is in the set of strings output by $A'(x)$ we are guaranteed that at least one string will not be removed from the set E. And, our assumption that A' outputs less than $2^{|x|}$ strings (and recall $2^{|x|}$ is the length of the strings) ensures that at some point during the computation, the i^{th} bit of all strings left in E will agree. Indeed, with each iteration of the for loop, if some strings disagree on the i^{th} bit position, at least one of them will be removed. Since, this can occur $2^{|x|}-2$ times and there are at most $2^{|x|} - 1$ strings in the set, we are guaranteed that at some point all

the strings will agree on the i^{th} bit position.

Note that the machine M' is able to compute the i^{th} bit of $A^+(x)$ using space $2^{|x|+i-2}$. Indeed, for each $0 \leq k < i$, M' simulates $A(s(x,k))$ which requires at most space $2^{|x|+k-2}$, and between each simulation the work tape is cleared for reuse.

To see that M' requires space at least $2^{|x|}$ for an infinite number of x, note that we can use the machine M to compute $A(x)$ for an infinite number of x. If M' used less space than $2^{|x|}$, we would be able to calculate the output of A for infinitely many x using less space than the space hierarchy theorem requires. □

3.8.5 Zig-zag assembly systems

In [31], a system $\mathcal{T} = (T, \sigma, \tau)$ is called a zig-zag tile assembly system provided that (1) $\mathcal{T}$ is directed, (2) there is a single sequence $\vec{\alpha} \in \mathcal{T}$ with $\mathcal{A}_{\square}[\mathcal{T}] = \{\vec{\alpha}\}$, and (3) for every $\vec{x} \in \text{dom } \alpha, (0,1) \notin \text{IN}^{\vec{\alpha}}(\vec{x})$. We say that an assembly sequences satisfying (2) and (3) is a *zig-zag assembly sequence*. Intuitively, a zig-zag tile assembly system is a system which grows to the left or right, grows up some amount, and then continues growth again to the left or right. Again, as defined in [31], we call a tile assembly system $\mathcal{T} = (T, \sigma, \tau)$ a *compact zig-zag tile assembly system* if and only if $\mathcal{A}_{\square}[\mathcal{T}] = \{\vec{\alpha}\}$ and for every $\vec{x} \in \text{dom } \alpha$ and every $\vec{u} \in U_2$, $\text{str}_{\alpha(\vec{x})}(\vec{u}) + \text{str}_{\alpha(\vec{x})}(-\vec{u}) < 2\tau$. Informally, this can be thought of as a zig-zag tile assembly system which is only able to travel upwards one tile at a time before being required to zig-zag again. The assembly sequence of a compact zig-zag system is called a *compact zig-zag assembly sequence*. Figure 3.8 depicts a compact zig-zag assembly sequence. As in the definition of a zig-zag system and throughout this section, we assume that each row of a zig-zag systems binds to the north of the previous row.

3.8.6 Space complexity of zig-zag systems is invariant under simulation

In this section, we give a formal definition of a language defined by a zig-zag system. We next show that such a language can be computed in space on the order of the maximal width of the zig-zag assembly grown to a finite height. While this result is fairly straight-

forward, we include it for the sake of completeness and because it serves as a basic example of how we will prove the main result of this section (Lemma 9). We give a formal definition of a language defined by a simulation of a zig-zag system, and Lemma 9 states that such a language can be computed in space on the order of the maximal width of the zig-zag assembly grown to a finite height.

Here is some of the notation used in this section. Let $\mathcal{T} = (T, \sigma, \tau)$ be a temperature τ compact zig-zag system with a seed σ consisting of a single tile, and let α be an assembly in $\mathcal{A}[\mathcal{T}]$. Since all of the results in this section hold regardless of the location of σ, without loss of generality, throughout this section, we assume that the location of σ is $(0, 0)$. Finally, we will use the term *configuration* to denote a partial function from a finite domain in $\mathbb{Z}^2$ to T, and *finite configuration* when the domain of the partial function from $\mathbb{Z}^2$ to T is finite.

Computational complexity and zig-zag systems

Let $T_1 \subseteq T$ be a subset of T, and let $r : \mathbb{N} \to \{0, 1\}$ be the function defined as

$$r(n) = \begin{cases} 1 & (0, n) \in \text{dom } \alpha \text{ and } \alpha((0, n)) \in T_1 \\ 0 & \text{otherwise.} \end{cases}$$

Now, let $f : \mathbb{N} \to \mathbb{N}$ be the function

$$f(n) = \max\{w_j \mid w_j \text{ is the width of the } j^{th} \text{ row of } \alpha \text{ for } 0 \leq j \leq n\}.$$

Finally, let $L_r = \{n \in \mathbb{N} \mid r(n) = 1\}$. We call r the *characteristic function for* $\mathcal{T}$ given T_1, and L_r the *language defined by* $\mathcal{T}$ *given* r. Notice that r is a computable function, f is a proper function, and L_r is a computable set. See Figure 3.8 for a description of how $r(n)$ is computed.

The following lemma gives an upper bound on the space complexity of a language defined by a zig-zag system.

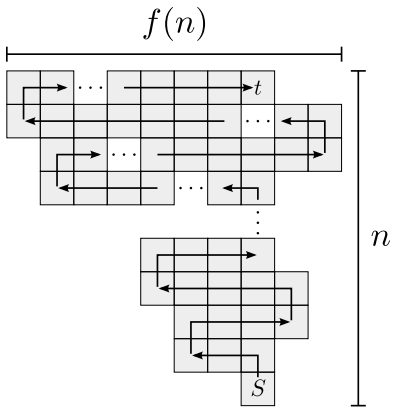


Figure 3.8: An assembly with a zig-zag assembly sequence. The assembly sequence is indicated with arrows. The tile labeled S makes up the seed σ, and $r(n) = 1$ if and only if the tile type of the tile labeled t is in T_1.

Lemma 6. Let $\mathcal{T} = (T, \sigma, \tau)$ be a zig-zag system with tile set T, seed assembly σ, and temperature τ. Let T_1 be some subset of T, let r be the characteristic function for $\mathcal{T}$ given T_1, and let L_r be the language defined by $\mathcal{T}$ given r. Finally, let $f(n)$ denote the width of the longest row of the assembly of $\mathcal{T}$ consisting of n completed rows. Then, $L_r \in \mathsf{DSPACE}(f(n))$.

Proof. Algorithm 5 defines steps for computing $r(n)$.

For an input $n \in \mathbb{N}$, notice that Algorithm 5 decides if n belongs to L_r. Notice that to perform this algorithm, as the top row of α assembles, the j^{th} row say, only the top two non-empty rows of the zig-zag assembly are needed. In other words, there is a computation that only requires space on the order of the tiles with locations $(i, j-1)$ or (i, j) for some $i \in \mathbb{Z}$. This is due to the fact that for zig-zag systems, once a tile is added to the j^{th} row, only tiles in these locations have glues that allow for the binding of an additional tile. There are at most $2f(n)$ tiles with locations $(i, j-1)$ or (i, j) for some $i \in \mathbb{Z}$ by the definition of f, and hence at most $2|T|f(n)$ tiles are required to compute $r(n)$. Therefore, $L_r \in \mathsf{DSPACE}(f(n))$. □

There are many generalities of Lemma 6 that can be made at this point. In the lemma, r is defined in terms of α, a compact zig-zag system, such that each row of α assembles

Input: $n \in \mathbb{N}$
Output: $r(n)$
$\alpha := \sigma$;
while $\partial^\tau \alpha \neq \varnothing$ **do**
  choose $(i, j) \in \mathbb{Z}^2$ such that $(i, j) \in \partial^\tau \alpha$;
  if $j = n + 1$ **then**
    break;
  end
  $l := (i, j)$;
  choose $t \in T$ such that $l \in \partial^\tau_t \alpha$;
  $\alpha := \alpha + (l \mapsto t)$;
end
if $(0, n) \in$ dom (α) *and* $\alpha((0, n)) \in T_1$ **then**
  return 1;
else
  return 0;
end

Algorithm 5: An algorithm for computing $r(n)$.

to the north of the previous row. We could just as easily have defined a compact zig-zag system so that each row assembles to the south, east, or west of a previous row. Also, note that it is not necessary that α is an assembly in a zig-zag system. In fact, $\mathcal{T}$ could be any TAS. It is only necessary that α in $\mathcal{A}[\mathcal{T}]$ has a compact zig-zag assembly sequence. Finally, the function r can also be generalized. r is defined using a single tile location, namely $(0, n)$, and the output of r is determined by $\alpha((0, n))$ and a subset of tile types T_1. Lemma 6 also holds if we define r to be defined using any finite number of tile locations and a set of configurations.

Paths in the binding graph

Before we proceed with the proof of Lemma 8, we give a way to find special paths in the binding graph of assemblies. Let $\mathcal{S} = (S, \sigma_S, \tau')$ be a TAS and let α be in $\mathcal{A}[\mathcal{S}]$. Let S_1 and S_2 be some nonempty finite sets of nonempty finite subassemblies in α. We say that S_1 and S_2 are *pairwise nonoverlapping* if for any pair of subassemblies $\beta_1 \in S_1$ and $\beta_2 \in S_2$, dom $(\beta_1) \cap$ dom $(\beta_2) = \varnothing$. Moreover, we say that S_2 *requires* S_1 if for any assembly sequence $\vec{\alpha}$ with result α, there exists an assembly α_1 in $\vec{\alpha}$ such that some subassembly of S_1

is a subassembly of α_1 and no subassembly of S_2 is a subassembly of α_1. Less formally, S_2 requires S_1 if at least one subassembly in S_1 must completely assemble prior to the assembly of any subassembly in S_2. Now we can state the following lemma about finding paths in the binding graph of an assembly.

Lemma 7. Let α be a stable finite assembly with an arbitrary valid assembly sequence $\vec{\alpha}$, and let S_1 and S_2 be nonempty finite sets of nonempty finite subassemblies of α such that 1) S_1 and S_2 are pairwise nonoverlapping and 2) S_2 requires S_1. Also, let α_1 be the first assembly in the assembly sequence $\vec{\alpha}$ containing any assembly of S_1, and let α_0 be the assembly in $\vec{\alpha}$ that immediately proceeds α_1 in $\vec{\alpha}$. Then there exists a path p in the binding graph of α with vertex set $V \subseteq \mathbb{Z}^2$ such that

1. for some $\gamma_1 \in S_1$ and $\gamma_2 \in S_2$, $V \cap \text{dom}\ (\gamma_1) \neq \varnothing$ and $V \cap \text{dom}\ (\gamma_2) \neq \varnothing$, and
2. $V \cap \text{dom}\ (\alpha_0) = \varnothing$

Proof. Since α is a stable assembly, it is clear that at least one path p in the binding graph of α satisfying Property 1 always exists. That we can find such a path satisfying Property 2 follows from the assumption that S_2 requires S_1. □

The path p in Lemma 7 corresponds to a path of tiles in α, and we call such a path the *a new path of tiles connecting* S_1 and S_2 to emphasize Property 2. Property 2 essentially says that such a path p forms only after a configuration in S_1 has assembled. To help motivate Lemma 7, consider that for a simulation $\mathcal{S}$ of a zig-zag system $\mathcal{T}$, in order for $\mathcal{S}$ to capture the dynamics of $\mathcal{T}$ correctly, for a row of macrotiles simulating a row of zig-zag growth from left to right from a tile t_1 to a tile t_2 say (respectively right to left), the set S_1 of macrotile subassemblies that represent t_1 requires the set S_2 of macrotile subassemblies that represent t_2. By Lemma 7, this implies that there is a new path of tiles connecting S_1 and S_2. In the proof of Lemma 8, we will use Lemma 7 to limit the amount of space that can be used as assembly in the simulating system proceeds.

Simulations of zig-zag systems

Let $\mathcal{S} = (S, \sigma_S, \tau')$ be a TAS that simulates $\mathcal{T}$ with representation function R and scale factor c. Since all of the results here hold upto translation of assemblies in $\mathbb{Z}^2$, without loss of generality, throughout this section we assume that the bottom-right tile of $\sigma_{\mathcal{T}}$ has location $(0, 0)$. Also let α' in $\mathcal{A}[\mathcal{S}]$ and α in $\mathcal{A}[\mathcal{T}]$ be assemblies such that $R^*(\alpha') = \alpha$. Furthermore, let $\vec{\alpha'}$ be an assembly sequence with result α'. Here we give a similar result to Lemma 6 for systems such as $\mathcal{S}$ that simulate zig-zag systems.

First, we introduce some notation similar to the notation used for stating Lemma 6. Let α'_n denote the subassembly of α' such that for all $(i, j) \in \text{dom } (\alpha'_n)$, $j \leq n$ and for all $(i, j) \in \partial^{\tau'} \alpha'_n$, $j \geq n + 1$. For some $L \subset \mathbb{Z}^2$ such that $|L| < \infty$ and for $\vec{v} \in \mathbb{Z}^2$, let $L_{\vec{v}}$ denote $\{\vec{l} + \vec{v} \mid \vec{l} \in L\}$. Also, for $\vec{n} = (0, n)$, let $C_n \subseteq \{w \mid w : L_{\vec{n}} \to S \text{ is a partial function}\}$ be a subset of configurations over S with domain in $L_{\vec{n}}$. Then we let $r' : \mathbb{N} \to \{0, 1\}$ be the function

$$r'(n) = \begin{cases} 1 & \alpha'_n|_{L_{\vec{n}}} \in C_n \\ 0 & \text{otherwise.} \end{cases}$$

Essentially, r' is the function obtained by growing α' to the point where the next tile added must be at a location above the line $y = n$, and then considering some finite configuration of this assembly. $r'(n) = 1$ if and only if this configuration is in C_n. Furthermore, let $f' : \mathbb{N} \to \mathbb{N}$ be the function defined as the maximum width w such that $w = |x_1 - x_2|$ where (x_1, y_1) and (x_2, y_2) are locations of some tiles in α' such that $0 \leq y_1, y_2 \leq n$. Finally, let $L'_r = \{n \in \mathbb{N} \mid r'(n) = 1\}$. As with zig-zag systems, we call r' the *characteristic function for $\mathcal{S}$ given C_n* and $L_{r'}$ the *language defined by $\mathcal{S}$ given r' and C_n*. Notice that r' is a computable function and $L_{r'}$ is a computable set. Lemma 8 gives an upper bound on the space complexity of a language defined by a system that simulates a zig-zag system.

Lemma 8. Let $\mathcal{T} = (T, \sigma, \tau)$ be a zig-zag system and let $\mathcal{S} = (S, \sigma_{\mathcal{T}}, \tau')$ be a directed system that simulates $\mathcal{T}$ with terminal assembly α'. Moreover, let L be a finite subset of $\mathbb{Z}^2$ and let C_n be a set of finite configurations over S with domain in $L_{\vec{n}}$, r' be the character-

istic function for $\mathcal{S}$ given C_n. Finally, $L_{r'}$ the language defined by $\mathcal{S}$ given r' and C_n, and let $f'(n)$ be the maximum width w such that $w = |x_1 - x_2|$ where (x_1, y_1) and (x_2, y_2) are locations of some tiles in α' such that $0 \leq y_1, y_2 \leq n$. Then, $L_{r'} \in \mathsf{DSPACE}(f'(n))$.

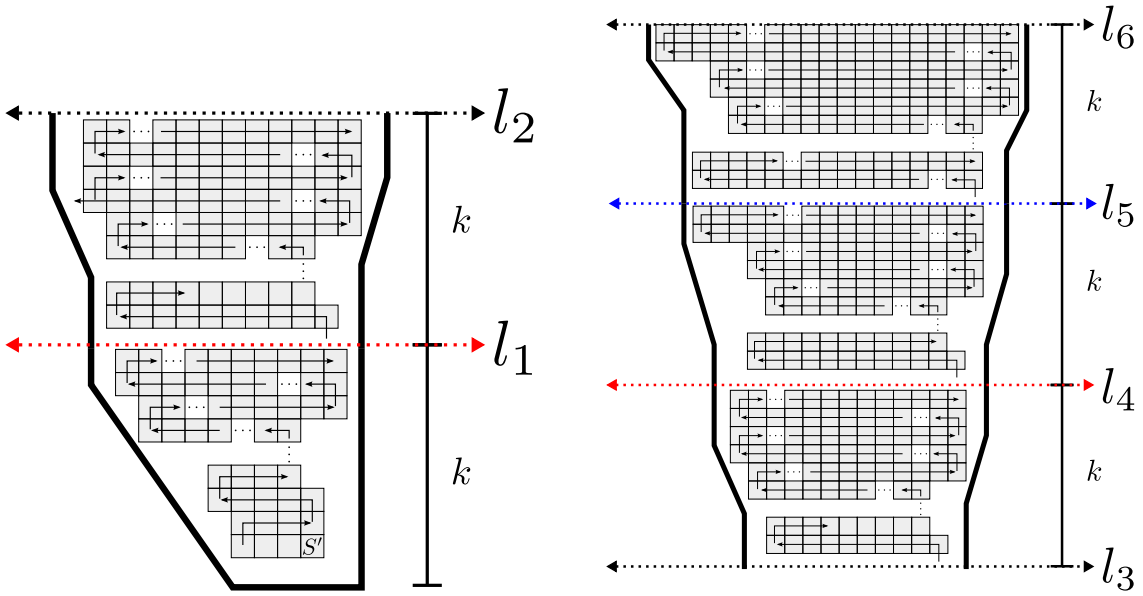
Overview of the proof of Lemma 8

In order to prove Lemma 8, we show that $r'(n)$ can be computed using space in $O(f'(n))$. Let c denote the scale factor of the simulation of $\mathcal{T}$ by $\mathcal{S}$, and let h denote the height of the smallest rectangle bounding L. We let $k = \max\{c^2 + 2c + 2, h\}$. As assembly proceeds from $\sigma_{\mathcal{T}}$, we take note of two regions where tiles may bind. First, tiles may bind in macrotile regions representing leftmost or rightmost tiles of the simulated zig-zag system or fuzz regions to the left or right of these macrotile regions. We call these macrotile regions the *left or right sides* of an assembly of $\mathcal{S}$. The second region where tiles may bind is the complement of the left or right sides of an assembly of $\mathcal{S}$.

The proof relies on a data structure called a *glue sequence table*[4]. This table, which we define later, is essentially a constant size (depending on U and c) lookup table that maps sequences of glues along a portion of a cut corresponding to the left or right sides of an assembly to sets of glues. The cut divides the $\mathbb{Z}^2$ lattice into two regions. One "above" the cut and one "below". The sets of glues mapped to by the table correspond to the glues that appear on the cut under the assumption that the sequence of glues are exposed along the cut in the order given by the sequence. Then, by using these exposed glues, tiles bind below the cut until there are no tile locations below the cut where a tile can stably bind. The glue sequence table has an entry for every possible glue sequence that can appear along the portion of the cut. Basically, the glue sequence table captures all "what-if" scenarios in the sense that it tells us what glues may be exposed along a cut by tiles binding below the cut if a glue (or a sequence of glues) is exposed along the cut by tiles located above the cut. See Figure 3.11 for a depiction of this process. We will describe the construction of a GST in the formal proof. It should be noted that it may not be possible for

[4]A glue sequence table is closely related to a *window-movie* (GST) as described in [29].

a sequence in the domain of a glue sequence table to correspond to an actual valid assembly.



(a) InitAssembly and InitGST (b) UpdateAssembly and UpdateGST

Figure 3.9: Determining configurations over $L_{\vec{n}}$.

In the proof, we describe four procedures for computing configurations over $L_{\vec{n}}$ starting from $\sigma_{\mathcal{T}}$. These procedures are 1) InitAssembly, 2) InitGST, 3) UpdateAssembly, and 4) UpdateGST. Figure 3.9 depicts a sketch of the assemblies formed by these procedures. Referring to Figure 3.9a, starting from the seed $\sigma_{\mathcal{T}}$, InitAssembly is the process of attaching all tiles that can stably bind at tile locations with y values $\leq 2k$. InitGST constructs the glue sequence table corresponding to cut shown in red by considering assemblies below the line that may form assuming that some glue (or sequence of glues) is exposed along the cut. Referring to Figure 3.9b, with the assembly below the line l_5 present, UpdateAssembly is the process of attaching all tiles that can stably bind a tile locations below l_6. This process uses the existing assembly between l_3 and l_5 and the glue sequence table corresponding to a cut shown in red. The result of UpdateAssembly is that the assembly below the the line l_6 and above the the line l_3 is terminal. Once UpdateAssembly is finished, UpdateGST constructs a new glue sequence table corresponding to a cut shown in blue. When the assembly below l_6 and above l_3 has formed and the glue sequence table has been

updated, the tiles below the line l_4 are no longer necessary for continuing the assembly of the simulation of the zig-zag system to the north. Using this fact, to finish the proof, we show that the four methods InitAssembly, InitGST, UpdateAssembly, and UpdateGST require space on the order of $f'(n)$ and can be used to compute $\alpha'_n|_{L_{\vec{n}}}$ in order to determine $r'(n)$.

Proof of Lemma 8

Proof. For $n \in \mathbb{N}$, we let H_n denote the set $\{(x, y) \in \mathbb{Z}^2 \mid y \leq n\}$. Based on Algorithm 6, we will give a means of computing $r'(n)$ that requires space in $O(f'(n))$.

Input: $n \in \mathbb{N}$
Output: $r'(n)$
`InitAssembly`;
`InitGST`;
while $\partial^{\tau'}\alpha' \cap H_n \neq \varnothing$ **do**
  `UpdateAssembly`;
  `UpdateGST`;
end
if $\alpha'(L_{\vec{n}}) \in C_n$ **then**
  return 1;
else
  return 0;
end

Algorithm 6: An algorithm for computing $r'(n)$.

The algorithm consists of four procedures that we describe next. Before we describe each of the procedures used in Algorithm 6, we introduce some notation. Let h denote the height of $L_{\vec{n}}$ and let $k = \max\{c^2 + 2c + 2, h\}$.

InitAssembly

We begin assembly in $\mathcal{S}$ by attaching tiles to σ_S until any tile that can stably bind to below the line $y = 2k$ has attached. We denote this step as the "InitAssembly" procedure. The InitAssembly procedure consists of the steps given in Algorithm 7.

Input: $\sigma_{\mathcal{T}}$
Output: α'_{2k}
$\alpha' := \sigma_S$;
while $\partial^{\tau'}\alpha' \cap H_{2k} \neq \varnothing$ **do**
  choose $(i,j) \in \mathbb{Z}^2$ such that $(i,j) \in \partial^{\tau'}\alpha'$ and $j \leq 2k$;
  $l := (i,j)$;
  choose $t \in T$ such that $l \in \partial_t^{\tau'}\alpha'$;
  $\alpha' := \alpha' + (l \mapsto t)$;
end

Algorithm 7: An algorithm describing the InitAssembly procedure.

InitGST

Before we describe the InitGST procedure, we define a data structure that is used in the algorithm called a *glue sequence table*. A glue sequence table is defined relative to a cut in the binding graph of $\alpha'_k \in \mathcal{A}[\mathcal{S}]$. Figure 3.10 depicts this cut, which we now describe.

To define the cut, we find two tiles such that the location of each of these tiles is above the line $y = k$. Let $i = \lceil k/c \rceil$ and let M_i^r be the macrotile region that maps to the tile, t_i^r say, in α that is farthest to the right on the i^{th} row α. Similarly, let M_i^l be the macrotile region that maps to the tile, t_i^l say, in α that is farthest to the left on the i^{th} row α. Moreover, let S_i^r (respectively S_i^l) be sets of configurations over S with domain in M_i^r (respectively M_i^i) such that each configuration of S_i^r (respectively S_i^l) maps to t_i^r (respectively t_i^l). Note that depending on the direction of growth (left or right) of the i^th row of the zig-zag system $\mathcal{T}$, either S_i^r requires S_i^l or S_i^l requires S_i^r. In either case, there must be a new path p_i of tiles from a tile of a configuration in S_i^r to a tile of a configuration in S_i^l. Note that for i between k and $2k$, each p_i is disjoint. This follows from Lemma 7. Now, p_i either contains a tile at a tile location that is below any tile of $\sigma_{\mathcal{T}}$ or it does not. The former case can only occur fewer than $2c$ times. This follows from the fact that if there were $2c$ or more such paths p_i, then these paths must contain tiles outside of any valid macrotile region representing a tile as well as outside of any fuzz regions of such macrotiles. In the latter case, there must be two tiles contained in M_i^r and M_i^l, which we denote by tl and tr respectively, such that these tiles are connected by a path of tiles in α'_{2k} and this path does

not contain a tile at a location below any tile of $\sigma_{\mathcal{T}}$. We will use the locations of tl'_i and tr'_i to define a cut of the grid graph.

Let (x_r, y_r) be the location of tr and let (x_l, y_l) be the location of tl. For the assembly α'_{2k} produced by the InitAssembly procedure, the cut, f_k say, is defined by the edges in the grid graph that intersects lines

1. $y = y_l + \frac{1}{2}$ for x in $(-\infty, x_l)$
2. $x = x_l + \frac{1}{2}$ for y between y_l and y_r
3. $y = y_r + \frac{1}{2}$ for x in (x_l, x_r)
4. $y = y_r + \frac{1}{2}$ for x in (x_r, ∞)

At a high-level, we have chosen this cut so that as assembly proceeds, tiles must either cross the portion of the cut corresponding to Lines 1 and 4 or are prevented from growing lower than all of the tiles of the path of tiles from tl to tr divide the plane into two disjoint sets. This idea is depicted in Figure 3.10.

We then denote the glue sequence table associated to the cut f_k by GST_{f_k}. Note that this cut extends infinitely to the left and right dividing the $\mathbb{Z}^2$ lattice into points "above" f_k and points "below" f_k. Analogously, for $j \in \mathbb{N}$, f_{jk} is defined as in f_k so that f_{jk} is a cut corresponding to lines that lie between the lines $y = ik$ and $y = (i+1)k$.

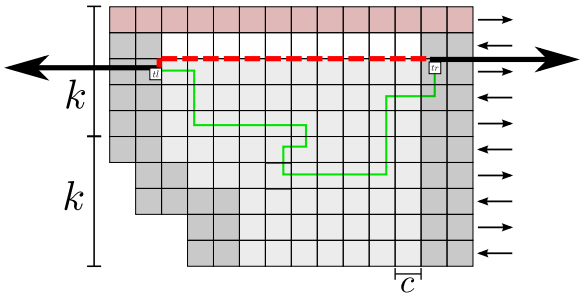


Figure 3.10: The cut used to define a GST. Glues which cross the red dotted portion of the cut are not in the domain of the GST. The green path is a representation of a path of tiles from tl to tr.

Let G be the set of glues associated to tiles in S and let $D = \{(0,1),(0,-1),(1,0),(-1,0)\}$ be a set corresponding to the north, south, east, and west edges of a tile respectively. In addition, let $\tilde{G}$ denote the product $G \times D \times \mathbb{Z}^2$ such that (t,d,l) is in $\tilde{G}$ iff $l = (x,y)$ where $y - \frac{1}{2}$ is not a point on Line 2 or 3. We say that $\tilde{g} \in \tilde{G}$ *crosses the cut* f_k iff $\tilde{g} = (g,d,l)$ for $g \in G$, $d \in D$, and $l \in \mathbb{Z}^2$ such that l lies on one side of the cut f_k and $l + d$ lies on the other side of the cut f_k. Moreover, if l is above (respectively below) the cut, we say that $\tilde{g}$ *crosses the cut from above to below* (respectively *crosses the cut from below to above*). Also, we take the phrase $\tilde{g}$ *is present* to mean that for $\tilde{g} = (g,d,l)$ for $g \in G$, $d \in D$, and $l \in \mathbb{Z}^2$, there is a tile at location l with glue g exposed on its edge corresponding to d.

Then, GST_{f_k} is a relation between the set of all sequences of length at most $4c$ over $\tilde{G}$ to the power set $\mathcal{P}\left(\tilde{G}\right)$. For a sequence of glues $\Sigma = \langle \tilde{g}_i \rangle_{0 \leq i < 4c, \tilde{g}_i \in \tilde{G}}$ and a set F in $\mathcal{P}\left(\tilde{G}\right)$, a pair (Σ, F) is in GST_{f_k} if and only if each $\tilde{g}_i$ in Σ crosses the cut f_k from above to below and F is the set of elements of $\tilde{G}$ obtained as follows. First, assume that the subassembly below the cut f_k is terminal and that the set F is empty. Now, consider assembly below the cut while assuming that $\tilde{g}_0$ is present. Note that it may not even be possible for a tile to be part of a valid assembly so that $\tilde{g}_0$ is present. With $\tilde{g}_0$ present, attach tiles at tile locations below the cut f_k. $\tilde{g}_0$ must be used to start tile attachment below the cut f_k. Continue attaching tiles until the newly assembled subassembly below the cut f_k is terminal. Call the new configuration of tiles β'_0. Then add any glues crossing the cut f_k from below to above to the set F. See Figure 3.11 for more detail. Now, assume that $\tilde{g}_1$ is present and attach tiles to β'_0 below the cut f_k until the produced assembly below the cut f_k is terminal. Add any new glues of this assembly that are crossing the cut f_k from below to above to the set F. Continue this process for each $\tilde{g}_i$ in Σ to construct the set F. This process is described in Algorithm 8. In this algorithm, we use the notation $\partial^{\tau'}(\alpha' \cup \tilde{g})$ to denote empty tile locations where, under the assumption that $\tilde{g}$ is present, exposed glues of α' and $\tilde{g}$ allow for a tile to be placed so that the sum of the glue strengths of glues of this tile that

match exposed glues of α' and/or $\tilde{g}$ is greater than or equal to τ'. It should be noted that in the following algorithm, α' does not necessarily denote a *stable* assembly; it only denotes a configuration of tiles.

Input: α'_{2k}
Output: GST_{f_k}
$\alpha' := \alpha'_{2k}$;
set Σ^* to the set of all sequence over $\tilde{G}$;
set B to the set of all $(i,j) \in \mathbb{Z}^2$ below f_k;
for $\Sigma \in \Sigma^*$ **do**
  set F to the empty set;
  set E to the empty set;
  for $\tilde{g} \in \Sigma$ **do**
    assume $\tilde{g}$ is present;
    while $B \cap \partial^{\tau'}(\alpha' \cup \tilde{g}) \neq \varnothing$ **do**
      choose $(i,j) \in \mathbb{Z}^2$ such that $(i,j) \in B \cap \partial^{\tau'}(\alpha' \cup \tilde{g})$;
      $l := (i,j)$;
      choose $t \in T$ such that $l \in \partial_t^{\tau'}(\alpha' \cup \tilde{g})$;
      $\alpha' := \alpha' + (l \mapsto t)$;
      $\tilde{b} := (t,d,l)$;
      if $\tilde{b}$ *crosses the cut* f_k *from below to above* **then**
        add $\tilde{b}$ to E;
      end
    end
  end
  set F to $F \cup E$;
  add (Σ, F) to GST_{f_k};
end

Algorithm 8: An algorithm describing the procedure InitGST.

UpdateAssembly

Let B_i be the set of all $(x,y) \in \mathbb{Z}^2$ such that $(i-2)k \leq y \leq ik$. Moreover, let β'_{ik} be the subconfiguration of α'_{ik} contained in B_i. In other words, β'_{ik} is the map $\alpha'_{ik}|_{B_i}$. The UpdateAssembly procedure consists of the steps given in Algorithm 9. This algorithm computes β'_{ik} using $\beta'_{(i-1)k}$ and $GST_{f_{(i-2)k}}$. The idea is to assemble the portion of β'_{ik} by allowing tiles to bind to tiles of $\beta'_{(i-1)k}$ when appropriate glues are present for strength two binding. In addition, when a tile is placed so that a glue, g say, on the tile crosses the cut

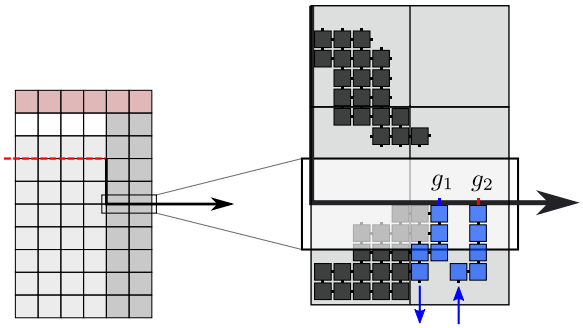


Figure 3.11: Construction of a glue sequence table. The bold line along with the red dotted line depict the cut f_k. Dark grey tiles are part of the existing assembly. The subassembly below the cut f_k is assumed to be terminal. If the blue glue g_1 is present, then assembly of the blue tiles may be possible resulting in glue g_2 crossing the cut f_k from below to above. This glue is added to the set F.

f, rather than continue to attach tiles below the cut using this exposed glue, the table $GST_{f_{(i-2)k}}$ is used to lookup which glues will cross the f from below to above as a result of g.

It still remains to be shown that this algorithm correctly yields β'_{ik}. To see this, we prove the following claim.

Claim 5. For tiles a and b in α'_{ik} but not in $\alpha'_{(i-1)k}$ such that the location (x_1, y_1) of a is above the line $y = (i-1)k$ and b is a tile belonging to a macrotile such that the location (x_2, y_2) of b is not contained in a macrotile belonging to the left or right side regions of α'_{ik}, it must be the case that $|y_2 - y_1| < c^2 + 2c + 2$.

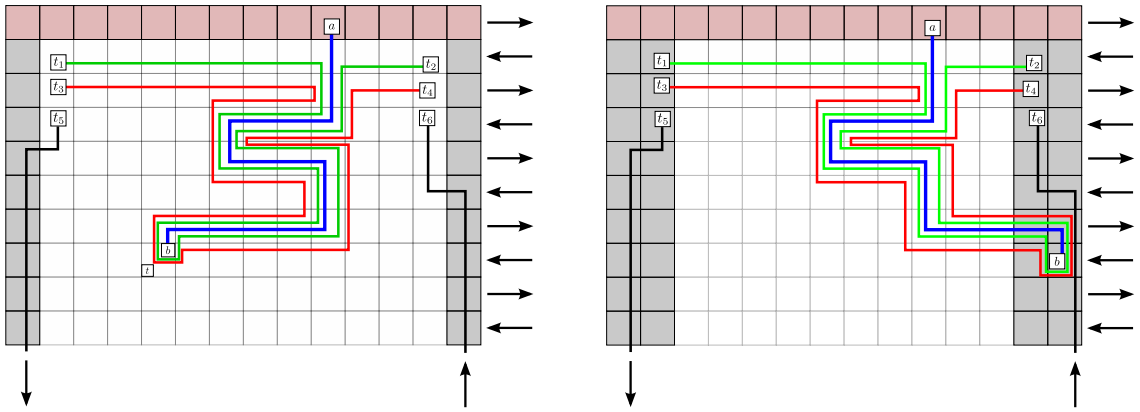
Note that a requires b. That is, b must be placed prior to a in any assembly sequence. By Lemma 7 there is a path of tiles from a to b. Let p_{ab} denote this path. Tiles a and b as described in Claim 5 are depicted in Figure 3.12. To prove the claim, consider the two tiles on the left and right ends of each row in the simulated zig-zag system. Call the tile farthest to the left t_1 and the other t_2. For the macrotile regions M_1 and M_2, where M_1 is to the left of M_2, of the simulating system that map to the tiles t_1 and t_2 respectively. We define S_1 to be the set of configurations over M_1 that map to t_1 under R^*. Similarly, we define S_2 to be the set of configurations over M_2 that map to t_2 under R^*. Note that for $\mathcal{S}$

Input: $\beta'_{(i-1)k}$ and $GST_{f_{(i-2)k}}$
Output: β'_{ik}
$\alpha' := \beta'_{(i-1)k}$;
set Σ to the empty sequence over $\tilde{G}$;
set B to the set of $(x, y) \in \mathbb{Z}^2$ such that $(i-2)k \leq y \leq ik$;
while $\partial^{\tau'}\alpha' \cap B \neq \varnothing$ **do**
    choose $(i, j) \in \mathbb{Z}^2$ such that $(i, j) \in \partial^{\tau'}\alpha' \cap B$;
    $l := (i, j)$;
    choose $t \in T$ such that $l \in \partial_t^{\tau'}\alpha'$;
    $\alpha' := \alpha' + (l \mapsto t)$;
    for $d \in \{(0, -1), (1, 0), (-1, 0)\}$ **do**
        $\tilde{g} := (t, d, l)$;
        if $\tilde{g}$ *crosses the cut* $f_{(i-2)k}$ *from above to below* **then**
            add $\tilde{g}$ to Σ as last element;
            assume that any unexposed glues in $GST_{f_{(i-2)k}}(\Sigma)$ along the cut $f_{(i-2)k}$ are present;
        end
    end
end
$\beta'_{ik} = \alpha'|_B$;

Algorithm 9: An algorithm describing the procedure UpdateAssembly.

to be a valid simulation of $\mathcal{T}$, either S_2 requires S_1 or S_1 requires S_2. Assume that S_2 requires S_1 (the other case is similar). Lemma 7 implies that there is a new path p of tiles from S_1 to S_2. In Figure 3.12a and 3.12b, p is depicted as the green line. Note that since b is not a tile in $\alpha'_{(i-1)k}$, we can assume that the path p from S_1 to S_2 does not intersect p_{ab}. Now we consider two cases. First, as the path p assembles, a tile is placed below any tile of the seed $\sigma_{\mathcal{T}}$. Second, as the path p assembles, a tile does not bind at a tile location that is below any tile location of the seed $\sigma_{\mathcal{T}}$. In the first case, the path must grow down the left or right side regions of α'_{ik}, and by our choice of cut f_{ik}, this path must cross the cut. For a valid simulation, the maximum number of such paths that can assemble is $2c$, for otherwise the paths would exceed the fuzz region allowable in simulation. In the second case, it must be the case that the tiles belonging to the path p place a tile in the middle region below b. In this case, the most such paths that can assemble is c^2. This follows from the fact that each such path places tiles either in the macrotile region containing the location

of b or below the macrotile region containing b. c^2 such paths would prevent any tiles from being placed in the macrotile region containing b, however, this would contradict the fact that $\mathcal{S}$ is a valid simulation. Therefore, Claim 5 holds.



(a) In this case, the tile b is contained in a macrotile region that is not a subregion of the left or right side regions shown as gray blocks.

(b) In this case, the tile b is contained in a macrotile region that is a subregion of the left or right side regions.

Figure 3.12: The blue, green, and red paths represent non-overlapping paths tiles of tiles that must assemble by Lemma 7.

By Claim 5, we see that since we have chosen k to be larger than $c^2 + 2c + 2$, Update-Assembly can be used to determine the tiles of β'_{ik} from $\beta'_{(i-1)k}$ and $GST_{f_{(i-2)k}}$.

UpdateGST

For each $i < n/k$, the procedure UpdateGST is used to create a new glue sequence table relative to the cut f_{ik}. The UpdateGST procedure is similar to the InitGST procedure, except that $GST_{f_{ik}}$ is constructed with the help of $GST_{f_{(i-1)k}}$. This procedure is given as Algorithm 10.

The space complexity of the computation of $r'(n)$ is $O(f'(n))$

In this section, we prove two claims. First, that Algorithm 6 correctly computes $r'(n)$, and second, that Algorithm 6 can be computed in space $O(f'(n))$. We first argue that Algorithm 6 can be computed in space $O(f'(n))$. Algorithm 6 consists of four procedures: Ini-

Input: β'_{ik} and $GST_{f_{(i-2)k}}$
Output: $GST_{f_{(i-1)k}}$
$\alpha' := \beta'_{ik}$;
set Σ^* to the set of all sequence over $\tilde{G}$;
set B to the set of all $(x, y) \in \mathbb{Z}^2$ below f_{ik};
for $\Sigma \in \Sigma^*$ **do**
    set F to the empty set;
    set M to the empty set;
    for $\tilde{g} \in \Sigma$ **do**
        set Π to the empty set; assume that $\tilde{g}$ is present;
        while $B \cap \partial^{\tau'}(\alpha' \cup \tilde{g}) \neq \varnothing$ **do**
            choose $(x, y) \in \mathbb{Z}^2$ such that $(x, y) \in B \cap \partial^{\tau'}(\alpha' \cup \tilde{g})$;
            $l := (x, y)$;
            choose $t \in T$ such that $l \in \partial_t^{\tau'}(\alpha' \cup \tilde{g})$;
            $\alpha' := \alpha' + (l \mapsto t)$;
            $\tilde{b} := (t, d, l)$;
            if $\tilde{b}$ *crosses the cut* $f_{(i-1)k}$ *from below to above* **then**
                add $\tilde{b}$ to M;
            end
            for $d \in \{(0, -1), (1, 0), (-1, 0)\}$ **do**
                $\tilde{d} := (t, d, l)$;
                if $\tilde{d}$ *crosses the cut* $f_{(i-2)k}$ *from above to below* **then**
                    add $\tilde{d}$ to Π as last element;
                    expose any unexposed glues in $GST_{f_{(i-2)k}}(\Pi)$ along the cut $f_{(i-2)k}$;
                end
            end
        end
    end
    set F to $F \cup M$;
    add (Σ, F) to $GST_{f_{(i-1)k}}$;
end

Algorithm 10: An algorithm describing the procedure UpdateGST.

tAssembly, InitGST, UpdateAssembly, and UpdateGST. First, we make the following observation.

Observation 2. $|\{(x, y) \mid GST_{f_k}(x) = y\}|$ is bounded by a constant that only depends on c and $|U|$. We denote this constant by K_{GST}. In particular, if g is the number of glues of tiles of U, $(g + 1)^{4c}(4c)!$ is such a constant as this is the total number of sequences of g glues (plus the null glue) of length $4c$.

From Algorithm 7 and 8, it is clear that InitAssembly and InitGST each require $O(f'(n) + K_{GST})$ space. Moreover, since for all i, β'_{ik} is bounded in width by $f'(n)$ and in height by $2k$, UpdateAssembly and UpdateGST each require $O(f'(n))$ space, with each procedure requiring at most $f'(n) * 3k$ tile locations.

Now let B_i denote the set of $(x, y) \in \mathbb{Z}^2$ such that $(i - 2)k \leq y \leq ik$. It remains to be shown that Algorithm 6 correctly computes $r'(n)$. To see this, note that UpdateAssembly computes $\beta'_{ik} = \alpha'_{ik}|_B$ and $L_n \subseteq B$. It follows that, $\beta'_{ik}|_{L_n} = \alpha'_{ik}|_{L_n}$. Therefore, Algorithm 6 correctly computes $r'(n)$.

□

Note that in Lemma 8, the assumption that $\mathcal{S}$ is directed can be removed by defining the glue sequence table to map into the set of sets of glues corresponding to each possible set of glues that may cross the cut corresponding to this glue sequence table. This set of sets is still bounded by a constant depending on c and S. Fix an enumeration of this set of sets. Then, we modify the procedures UpdateAssembly and UpdateGST so that if a glue crosses the cut from above to below, we expose glues corresponding to the first set of glues in the enumeration of the set of sets of glues. Now we can state Lemma 9 which we refer to as the "no cheating lemma".

Lemma 9 (No Cheating Lemma)**.** Let $\mathcal{T} = (T, \sigma, \tau)$ be a zig-zag system and let $\mathcal{S} = (S, \sigma_S, \tau')$ be a system that simulates $\mathcal{T}$ at temperature τ' with scale factor c. Let n be in $\mathbb{N}$, and let $f(n)$ be the width of the longest row of the assembly of $\mathcal{T}$ consisting of n

completed rows. Moreover, let C_{cn} be a set of finite configurations, let r' be the characteristic function for $\mathcal{S}$ given C_{cn}, and let $L_{r'}$ be the language defined by $\mathcal{S}$ given r'. Then, $L_{r'} \in \mathsf{DSPACE}(f(n))$.

Proof. For the scale factor c, this follows from the fact that $f'(n) \leq cf(n) + 2c$. The addition of $2c$ accounts for fuzz regions. □

Chapter 4

A computationally universal, non-cooperative model

4.1 Introduction

A long standing open conjecture in regards to the aTAM is that systems in which tile attachments depend only on one exposed glue (we call such systems *temperature-1* systems) are not computationally universal [10, 29, 28]. It may appear clear that this conjecture is certainly true, but the ability of tile assembly systems to place a tile which prevents the attachment of a later tile gives these systems a surprising amount of power [15, 13, 20, 6] and has made proving such a result elusive. In fact, the exploitation of this ability has been used to show that temperature-1 systems in other models are computationally universal [31, 4, 15, 20, 13].

This paper examines the computational power of a model which is similar to the aTAM with the exception that the shape of the tiles in the systems is relaxed to include any shape which is a polygon. Unlike all previous work, our model makes no assumption about an underlying lattice and discrete space. Instead, we must work in the real plane, and fundamentally exploit continuous space to precisely position polygonal tiles. We call this model the polygonal TAM and show that certain classes of temperature-1 systems in the polygonal TAM are computationally universal. In order to show our results about computational universality, we explicitly construct "lattices" for polygons and create geometric "bit-readers". In the case of regular polygons with $n > 6$ sides, we exploit the inability of these polygons to tile the plane to read bits. In fact, we show that for regular polygons which do tile the plane, bit-reading gadgets are impossible to construct. Interestingly, our exploits do not work for pentagons. In particular, we show that even though pentagons cannot tile the plane, bit-reading gadgets are impossible to construct with them.

The layout of the paper is as follows. We first introduce the polygonal TAM. Next, we introduce our main results which concentrate on the computational power of polygonal TAM systems at temperature 1. Our first main result states that for any regular poly-

gon P with $n > 6$ sides, there exists a polygonal TAM system consisting of tiles of shape P which simulates any Turing machine on any input. We then provide evidence that this computational boundary is tight by showing that the class of polygonal TAM systems composed only of tiles of a single shape which is any regular polygon P with $n \leq 6$ sides cannot compute using any currently known techniques. On the other hand, we show that the class of polygonal TAM systems whose tiles are composed of any two regular polygons is capable of simulating any Turing machine on arbitrary input. We then show two positive results about computing with systems whose tiles have the shape of non-regular polygons with less than seven sides. In order to show these results we have two supporting sections. One shows how we can create a "lattice" in the plane out of any regular polygon. The other uses these "lattices" to connect together several components which "read bits".

4.2 Preliminaries

We now give a description of the Polygonal TAM[1] .

Polygonal Tiles A *simple polygon* is a plane geometric figure consisting of straight, non-intersecting line segments or "sides" that are joined pair-wise to form a closed path. As is commonly the case, we omit the qualifier "simple" and refer to simple polygons as polygons. A polygon encloses a region called its *interior*. The line segments that make-up a polygon meet only at their endpoints. Exactly two edges meet at each vertex. We define the set of *edges* of a polygon to be the line segments that make-up a polygon. In our definition we find it useful to give a polygon a default position and rotation. First, we assume that the centroid, c say, of any polygon is at the origin in $\mathbb{R}^2$. Then, for a polygon P_n with n edges, let $v = (x, y) \in \mathbb{R}^2$ be some vertex of P_n such that $v \neq c$. By possibly rotating P_n about c, we can ensure that $y = 0$ and $x > 0$. For a given polygon P and some vertex v of P that is not equal to the centroid of P, we call this position and rotation the *standard*

[1]The Polygonal TAM is simply a case of the polygonal free-body TAM defined in [6] with no rotational restriction and no tile flipping. We define it here for completeness.

position for P given v.

A *polygonal tile* is a polygon with a subset of its edges labeled from some *glue* alphabet Σ, with each glue having an integer *strength* value. Two tiles are said to be *adjacent* if they are placed so that two edges, one on each tile, intersect completely. Two tiles are said to *bind* when they are placed so that they have non-overlapping interiors and have adjacent edges with matching glues and matching lengths; each matching glue binds with force equal to its strength value. An *assembly* is any connected set of polygons whose interiors do not overlap such that every tile is adjacent to some other tile. [2] Given a positive integer $\tau \in \mathbb{N}$, an assembly is said to be τ-*stable* or (just *stable* if τ is clear from context), if any partition of the assembly into two non-empty groups (without cutting individual polygon) must separate bound glues whose strengths sum to $\geq \tau$. We say that a tile is in *standard position*, if the underlying polygon defining the shape of the tile is in standard position. We also refer to the centroid of a polygonal tile as the centroid of the underlying polygon defining the shape of the tile.

Tile System A *tile assembly system* (TAS) is an ordered triple $\mathcal{T} = (T, \sigma, \tau)$ where T is a set of polygonal tiles, and σ is a τ-stable assembly called the *seed*. τ is the *temperature* of the system, specifying the minimum binding strength necessary for a tile to attach to an assembly. Throughout this paper, the temperature of all systems is assumed to be 1, and we therefore frequently omit the temperature from the definition of a system (i.e. $\mathcal{T} = (T, \sigma)$). If the tiles in T all have the same polygonal shape, $\mathcal{T}$ is said to be a *single-shape* system; more generally $\mathcal{T}$ is said to be a c-shape system if there are c distinct shapes in T. If not stated otherwise, systems described in this paper should by default be assumed to be single-shape systems.

We define a *configuration* of $\mathcal{T}$ to be a (possibly empty) arrangement of tiles in $\mathbb{R}^2$ where tiles of this arrangement are translations and/or rotations of copies of tiles in T.

[2] As with the aTAM, the edges of two tiles of an assembly may intersect, but we do not allow for the interiors of two tiles of an assembly to have non-empty intersection.

Formally, we define a configuration of $\mathcal{T}$ as follows. For a c-shaped system $\mathcal{T} = (T, \sigma, \tau)$, let P_1, P_2, ..., P_c denote the polygons that make up the shapes of $\mathcal{T}$. For each i such that $1 \leq i \leq c$, assume that each P_i is in standard position given some vertex v_i of P_i. Then, a configuration of $\mathcal{T}$ is a partial function $\alpha : \mathbb{R}^2 \dashrightarrow T \times [0, 2\pi)$. One should think of this function as mapping centroid locations, r say, to a tile-angle pair (t, θ) with tile t in T and orientation angle θ as follows. Starting from t in standard position, t is rotated counter-clockwise by θ and translated so that the centroid of t is at r. Note that the definition of configuration makes no claim as to whether or not two tiles of a configuration have overlapping interiors or have matching glues. Similarly, we can define an assembly to be a configuration such that every tile is adjacent to some other tile and the intersection of the interiors of any two distinct tiles is empty. Then an assembly α' is a *subassembly* of α if dom $(\alpha') \subseteq$ dom (α) and if $(r, \theta) \in$ dom (α') then $\alpha((r, \theta)) = \alpha'((r, \theta))$. We define subconfiguration analogously to the way we defined subassembly.

Assembly Process Given a tile-assembly system $\mathcal{T} = (T, \sigma, \tau)$, we now define the set of *producible* assemblies $\mathcal{A}[\mathcal{T}]$ that can be derived from $\mathcal{T}$, as well as the *terminal* assemblies, $\mathcal{A}_{\square}[\mathcal{T}]$, which are the producible assemblies to which no additional tiles can attach. The assembly process begins from σ and proceeds by single steps in which any single copy of some tile $t \in T$ may be attached to the current assembly A, provided that it can be translated and/or rotated so that its placement does not overlap any previously placed tiles and it binds with strength $\geq \tau$. For a system $\mathcal{T}$ and assembly A, if such a $t \in T$ exists, we say $A \rightarrow_1^{\mathcal{T}} A'$ (i.e. A grows to A' via a single tile attachment). We use the notation $A \rightarrow^{\mathcal{T}} A''$, when A grows into A'' via 0 or more steps. Assembly proceeds asynchronously and nondeterministically, attaching one tile at a time, until no further tiles can attach. An *assembly sequence* in a TAS $\mathcal{T}$ is a (finite or infinite) sequence $\vec{\alpha} = (\alpha_0 = \sigma, \alpha_1, \alpha_2, \ldots)$ of assemblies in which each α_{i+1} is obtained from α_i by the addition of a single tile. The set of producible assemblies $\mathcal{A}[\mathcal{T}]$ is defined to be the set of all assemblies A such that there exists an assembly sequence for $\mathcal{T}$ ending with A (possibly in the limit). The set of *terminal* as-

semblies $\mathcal{A}_{\square}[\mathcal{T}] \subseteq \mathcal{A}[\mathcal{T}]$ is the set of producible assemblies such that for all $A \in \mathcal{A}_{\square}[\mathcal{T}]$ there exists no assembly $B \in \mathcal{A}[\mathcal{T}]$ in which $A \rightarrow_1^{\mathcal{T}} B$. A system $\mathcal{T}$ is said to be directed if $|\mathcal{A}_{\square}[\mathcal{T}]| = 1$, i.e., if it has exactly one terminal assembly.

4.3 Geometric bit-reading, grids, and Turing machine simulation

In this section we state our main results and then give a high-level description of the machinery used to prove these results. In particular, we describe bit-reading gadget assemblies and grid assemblies, and briefly show how to simulate a Turing machine using these assemblies. The general strategy that motivates the work in this paper is similar to the techniques used in [4, 20, 13]. Unlike the techniques used in [4, 20, 13], we do not have an underlying integer lattice that is being tiled, and therefore, must rely on analysis of polygonal tile assemblies in $\mathbb{R}^2$.

4.3.1 Main results

We now state our main results. The first set of results are positive and state that there are a variety of systems with polygons which can simulate any Turing machine. The last result is a negative result which states that the class of systems whose tiles are composed of regular polygons with less than 7 sides cannot compute using known techniques in self-assembly.

Informally, our first theorem states that if P is a regular polygon with ≥ 7 sides, then the class of systems with tiles of shape P is computationally universal.

Theorem 2. Let P_n be a regular polygon with n sides such that $n \geq 7$. Then for every standard Turing machine M and input w, there exists a directed TAS with $\tau = 1$ consisting only of tiles of shape P_n that simulates M on w.

The following theorem states that if we are allowed two different regular polygons as tile shapes, then the class of systems consisting only of these two shapes is computationally universal.

Theorem 3. Let P_n and Q_m be regular polygons with n and m sides of equal length. Then for every $n \geq 3$ and $m \geq 3$ such that $n \neq m$, and every standard Turing machine M with input w, there exists a directed 2-shaped system $\mathcal{T}_{n,m} = (T_{n,m}, \sigma_{n,m})$ consisting only of tiles of shape P_n or Q_m that simulates M on w.

The next theorem differs from the previous two theorems in that it discusses the computational power of polygons which are not regular. Roughly, it states that if we relax the condition that the polygon is regular (but still equilateral), then there exist polygons with only four sides which are capable of composing a class of computationally universal single shape systems. It also implies this for shapes with five and six sides as well.

Theorem 4. Let M be a standard Turing machine with input w. Then for all $n \geq 4$, there exists an equilateral polygon P_n with n sides and a directed single-shaped system $\mathcal{T}_n = (T_n, \sigma_n)$ consisting only of tiles of shape P_n that simulates M on w.

Our final positive result shows that there exists a class of single-shaped systems of obtuse isosceles triangle which is computationally universal.

Theorem 5. Let M be a standard Turing machine with input w. Then, there exists an obtuse isosceles triangle P and a directed single-shaped system $\mathcal{T} = (T, \sigma)$ consisting only of tiles of shape P that simulates M on w.

We now state the negative result, which is based on the fact that regular polygonal tiles with ≤ 6 sides cannot form paths capable of blocking each other in specific ways allowing important geometric information encoding and decoding.

Theorem 6. Let $n \in \mathbb{N}$ be such that $3 \leq n \leq 6$. Then, there exists no temperature 1 single-shaped polygonal tile assembly system $\mathcal{T} = (T, \sigma, 1)$ where for all $t \in T$, t is a regular polygon with n sides, and a bit-reading gadget exists for $\mathcal{T}$.

Due to space constraints in this extended abstract, the proofs of most results are relegated to the Appendix. However, in the main body we now sketch an overview of how the positive results work, and we provide the proof of the negative result in full.

4.3.2 Bit-reading gadgets overview

First, we discuss a primitive tile-assembly component that enables computation by self-assembling systems. This component is called the *bit-reading gadget*, and essentially consists of pre-existing assemblies, *bit writers*, that appropriately encode bit values (i.e., 0 or 1) and paths that grow past them and are able to "read" the values of the encoded bits; this results in those bits being encoded in the tile types of the paths beyond the encoding assemblies. The notion of bit-reading gadget was defined in [13]. For completeness, we present the definition here and note that the definition applies even to systems of polygonal tiles. Figure 4.1 provides an intuitive overview of a temperature-1 system with a bit-reading gadget. Essentially, depending on which bit is encoded by the assembly to be read, exactly one of two types of paths can complete growth past it, implicitly specifying the bit that was *read*. It is important that the bit reading must be unambiguous, i.e., depending on the bit *written* by the pre-existing assembly, exactly one type of path (i.e., the one that denotes the bit that was written) can possibly complete growth, with all paths not representing that bit being prevented. Furthermore, the correct type of path must always be able to grow. Therefore, it cannot be the case that either all paths can be blocked from growth, or that any path not of the correct type can complete, regardless of whether a path of the correct type also completes, and these conditions must hold for any valid assembly sequence to guarantee correct computation.

The key to the correct functioning of a bit-reading gadget at temperature-1, where glue cooperation is not available and one source of "input" to the growing bit-reader must instead be provided by geometry, in the form of geometric hindrance which prevents exactly one path from continuing growth but allows another to proceed, is the fact that it must work when reading either of two different bit values. Using Figure 4.1 as a guide, one can see that it is easy to read the "1" bit in this example by blocking the blue path. However, the difficulty which is encountered is in correctly blocking the yellow path while allowing the blue to continue in order to read a "0" bit. With square tiles (and as we show, several

others), this is in fact impossible. However, with most polygonal tiles this can be accomplished by careful design of paths and blocking assemblies so that a gap remains between the blocked path and the blocking assembly in such a way that the other path can assemble through the gap. The techniques for accomplishing this will be demonstrated throughout this paper.

Here and throughout the paper, if we refer to a tile having an x (or y) coordinate i, we are referring to its centroid being on the line $x = i$ (or $y = i$) for $i \in \mathbb{R}$.

Definition. We say that a *bit-reading gadget* exists for a tile assembly system $\mathcal{T} = (T, \sigma, \tau)$, if the following hold. Let $T_0 \subset T$ and $T_1 \subset T$, with $T_0 \cap T_1 = \varnothing$, be subsets of tile types which represent the bits 0 and 1, respectively. For some producible assembly $\alpha \in \mathcal{A}[\mathcal{T}]$, there exist two connected subassemblies, $\alpha_0, \alpha_1 \sqsubseteq \alpha$ (with w equal to the maximal width of α_0 and α_1, i.e., the largest extent in x-direction spanned by either subassembly), such that if:

1. α is translated so that α_0 has its minimal y-coordinate ≤ 0 and its minimal x-coordinate ≥ 0,

2. a tile of some type $t \in T$ is placed at $(w + n, h)$, where $n, h \geq 1$, and

3. the tiles of α_0 are the only tiles of α in the first quadrant to the left of t,

then at least one path must grow from t (staying strictly above the x-axis) and place a tile of some type $t_0 \in T_0$ as the first tile with x-coordinate < 0, while no such path can place a tile of type $t' \in (T \setminus T_0)$ as the first tile to with x-coordinate < 0. (This constitutes the reading of a 0 bit.)

Additionally, if α_1 is used in place of α_0 with the same constraints on all tile placements, t is placed in the same location as before, and no other tiles of α are in the first quadrant to the left of t, then at least one path must grow from t and stay strictly above the x-axis and strictly to the left of t, eventually placing a tile of some type $t_1 \in T_1$ as the

first tile with x-coordinate < 0, while no such path can place a tile of type $t' \in (T \setminus T_1)$ as the first tile with x-coordinate < 0. (Thus constituting the reading of a 1 bit.)

We refer to α_0 and α_1 as the *bit writers*, and the paths which grow from t as the *bit readers*. Also, note that while this definition is specific to a bit-reader gadget in which the bit readers grow from right to left, any rotation of a bit reader is valid by suitably rotating the positions and directions of Definition 4.3.2.

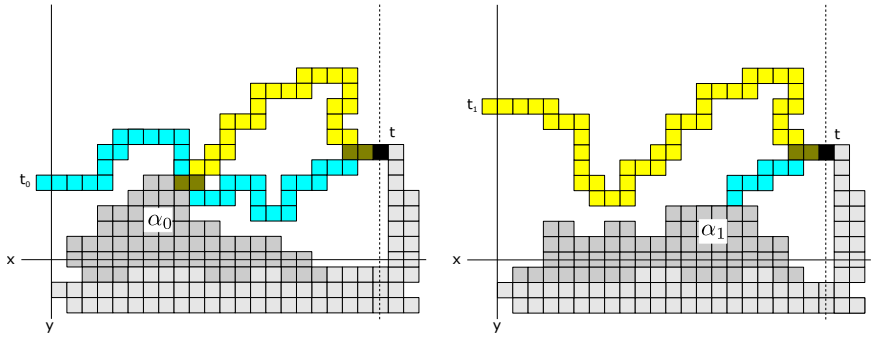


Figure 4.1: Abstract schematic of a bit-reading gadget. (Left) The blue path grown from t "reads" the bit 0 from α_0 (by being allowed to grow to $x < 0$ and placing a tile $t_0 \in T_0$), while the yellow path (which could read a 1 bit) is blocked by α_0. (Right) The yellow path grown from t reads the bit 1 from α_1, while the blue path that could potentially read a 0 is blocked by α_1. Clearly, the specific geometry of the used polygonal tiles and assemblies is important in allowing the yellow path in the left figure to be blocked without also blocking the blue path.

4.3.3 Formal definition of bit-reading gadget

The following definition is taken from [13] and modified slightly to account for the fact that polygonal tiles are placed in continuous, rather than discrete, space. Here and throughout the paper, if we refer to a tile having an x (or y) coordinate i, we are referring to its centroid being on the line $x = i$ (or $y = i$) for $i \in \mathbb{R}$.

Definition. We say that a *bit-reading gadget* exists for a tile assembly system $\mathcal{T} = (T, \sigma, \tau)$, if the following hold. Let $T_0 \subset T$ and $T_1 \subset T$, with $T_0 \cap T_1 = \varnothing$, be subsets of tile types which represent the bits 0 and 1, respectively. For some producible assembly

$\alpha \in \mathcal{A}[\mathcal{T}]$, there exist two connected subassemblies, $\alpha_0, \alpha_1 \sqsubseteq \alpha$ (with w equal to the maximal width of α_0 and α_1, i.e., the largest extent in x-direction spanned by either subassembly), such that if:

1. α is translated so that α_0 has its minimal y-coordinate ≤ 0 and its minimal x-coordinate ≥ 0,

2. a tile of some type $t \in T$ is placed at $(w + n, h)$, where $n, h \geq 1$, and

3. the tiles of α_0 are the only tiles of α in the first quadrant to the left of t,

then at least one path must grow from t (staying strictly above the x-axis) and place a tile of some type $t_0 \in T_0$ as the first tile with x-coordinate < 0, while no such path can place a tile of type $t' \in (T \setminus T_0)$ as the first tile with x-coordinate < 0. (This constitutes the reading of a 0 bit.)

Additionally, if α_1 is used in place of α_0 with the same constraints on all tile placements, t is placed in the same location as before, and no other tiles of α are in the first quadrant to the left of t, then at least one path must grow from t and stay strictly above the x-axis and strictly to the left of t, eventually placing a tile of some type $t_1 \in T_1$ as the first tile with x-coordinate < 0, while no such path can place a tile of type $t' \in (T \setminus T_1)$ as the first tile with x-coordinate < 0. (Thus constituting the reading of a 1 bit.)

We refer to α_0 and α_1 as the *bit writers*, and the paths which grow from t as the *bit readers*. Also, note that while this definition is specific to a bit-reader gadget in which the bit readers grow from right to left, any rotation of a bit reader is valid by suitably rotating the positions and directions of Definition 4.3.3.

4.3.4 Grid assemblies

As we will see in Section 4.3.5, our construction to simulate a Turing machine with a Polygonal TAM system consisting of the polygon P will require us to string together several bit writers which we will then read with a series of bit readers. In order to ensure that

the path which is assembling the bit readers is placing the bit readers at the correct positions, we need to keep track of where the bit writers are located. We accomplish this by constructing a lattice in the plane with P. We can then place our bit writers at periodic positions in this lattice so that the path which is assembling the bit readers will know where to place the bit readers.

4.3.5 Turing machine simulation

The text and figures in this section is taken from [13], which deals with polyomino shaped tiles. In order to show that a polygon shape (i.e., a system composed of tiles of only that shape) is computationally universal at $\tau = 1$, we show how it is possible to simulate an arbitrary Turing machine using such a polygon system, which is done in a manner logically identical to that described for polyminoes, allowing polygons to be substituted throughout the descriptions in this section. In order to simulate an arbitrary Turing machine, we show how to self-assemble a zig-zag Turing machine [4, 31]. A zig-zag Turing machine at $\tau = 1$ works by starting with its input row as the seed assembly, then growing rows one by one, alternating growth from left to right with growth from right to left. As a row grows across the top of the row immediately beneath it, it does so by forming a path of single tile width, with tiles connected by glues, which pass information horizontally through their glues, while the geometry of the row below causes only one of two choices of paths to grow at regular intervals, effectively passing information vertically via the geometry, using bit-reading gadgets.

Each cell of the Turing machine's tape is encoded by a series of bit-reader gadgets that encode in binary the symbol in that cell and, if the read/write head is located there, what state the machine is in. Additionally, as each cell is read by the row above, the necessary information must be geometrically written above it so that the next row can read it. See Figure 4.2 for an example depicting a high-level schematic without showing details of the individual polyominoes. Figure 4.3 shows the same system after two rows have completed growth.

Figure 4.2: High-level schematic view of a zig-zag Turing machine and the bit-reading/writing gadgets that make up each row of the simulation. The bottom shows the seed row, consisting of bit-writer gadgets separated by spacers. Then, depicted as consecutive upward figures, the second row begins its growth. Yellow/blue portions depict locations of bit-reader gadgets (for 0 and 1, respectively), which grow pink paths upward after completing in order to grow bit writer gadgets (grey), and then gold spacers back down to the point where the next bit reader can grow.

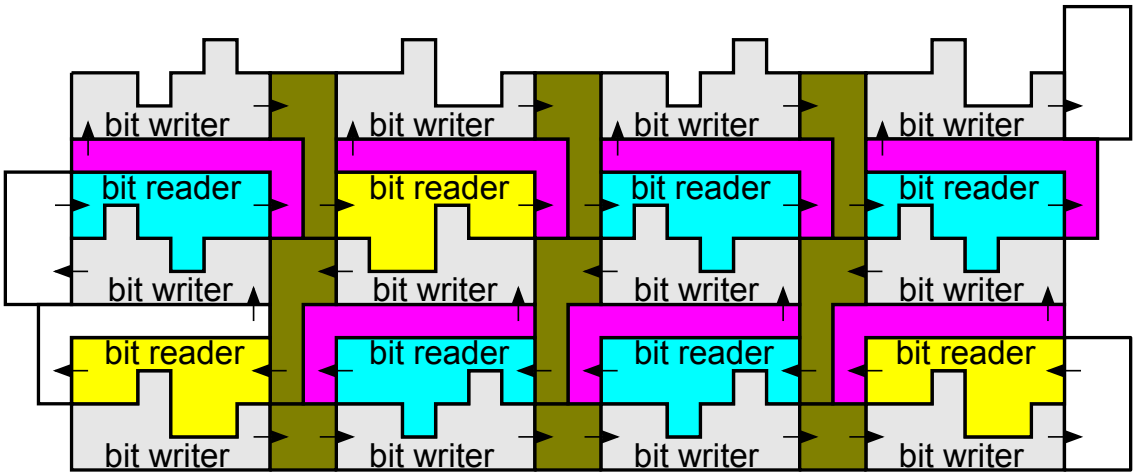


Figure 4.3: High-level schematic view of a zig-zag Turing machine and the bit-reading/writing gadgets that make up the first two rows of simulation.

For a more specific example that shows the placement of individual, actual polyomino tiles as well as the order of their growth, see Figure 4.4. Note that the simulation of a zig-zag Turing machine can be performed by horizontal or vertical growth, and in any orientation.

Thus, to show our positive results, our task has become to 1) show that bit reading gadgets exist for the claimed systems and 2) show that we can string them together. The first task is accomplished in Section 4.8 and the grid which allows us to show the latter is shown in Section 4.5.

Given an n-sided regular polygon P where $n > 6$, a Turing machine M and an input w, Algorithm 11 shows a high-level schematic view of an algorithm that produces a single shape Polygonal TAM system which simulates the Turing machine M on input w and consists of tiles of shape P. Note that here, we are abstracting the way in which the mathematical structures appearing in the algorithm are represented. In Section 4.5, we give a construction which implicitly defines an algorithm which we call FORM_GRID. This algorithm takes an integer n as input and returns a grid formed by the n-sided regular polygon. Given a grid $\mathcal{G}$ and an n-sided regular polygon, in Section 4.9 our construction implicitly gives an algorithm which we call FORM_GADGETS, that takes a grid $\mathcal{G}$ and an

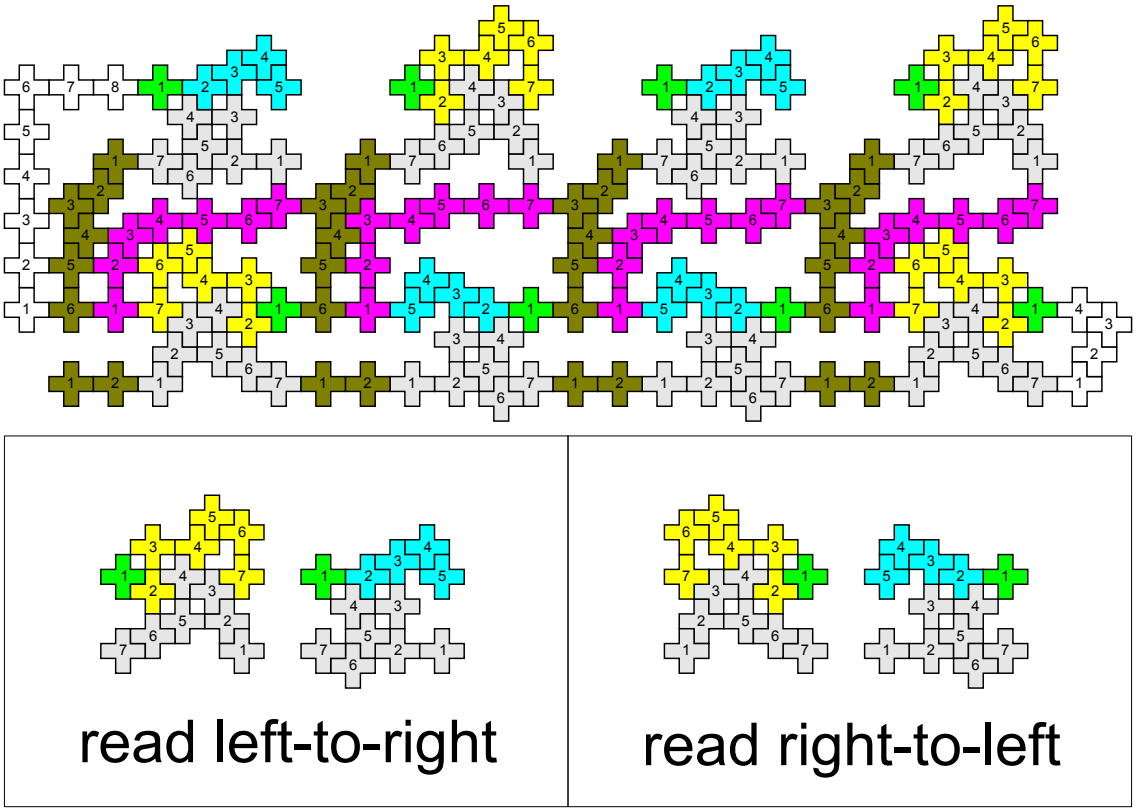


Figure 4.4: The system of Figure 4.2 after two rows of the zig-zag simulation have been completed (omitting the output bit writer gadgets of the second row), implemented with "plus-sign" polyominoes. The bottom left shows 0 and 1 bit-writer and reader combinations, with the writer having grown from right to left and the reader from left to right. The bottom right shows the same, but with growth directions reversed. Grey tiles represent bit-writer gadgets. Green tiles represent the beginning of bit-reader gadgets that are common to either bit; yellow represents the path that can grow to signify a 0 bit being read, and blue a 1 bit. Other colors correspond to those for the gadgets used in Figure 4.2, with numbers corresponding to the growth order of the tiles in each gadget.

integer n, and produces a normalized bit-reading gadget. Once we have a normalized bit reading gadget, we can use the algorithm implicitly described in Section 3.2 of [13], which we call INITIALIZE, that produces a system, say $\mathcal{T} = (T, \sigma)$, which grows a geometric representation of the input w. Finally, also in Section 3.2 of [13], an algorithm is implicitly given, which we call TRANSITION_TILES, that returns a set of tiles which are added to T so that the system $\mathcal{T}$ is able to simulate a transition of the Turing machines M.

Data: n, M, w
Result: Tile assembly system $\mathcal{T}$ which simulates M on w
$\mathcal{G} \leftarrow$ FORM_GRID(n);
$NRG \leftarrow$ FORM_GADGETS($\mathcal{G}$, n);
$\mathcal{T} = (T, \sigma) \leftarrow$ INITIALIZE(n, M, w, NRG);
$T \leftarrow T \cup$ TRANSITION_TILES(n, M, NRG);
return $\mathcal{T}$;

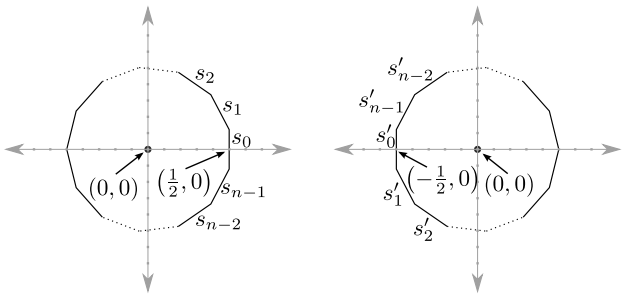
Algorithm 11: High level algorithm for constructing a system $\mathcal{T}$ which simulates M on w.

4.4 Regular polygonal tile analysis with complex roots

In order to construct the grid assemblies and to show the correctness of the bit-reading gadgets we must show that the grid configurations and the bit-reading gadget configurations result in a valid assembly. In other words, we must show that the intersection of the interiors of any two distinct polygonal tiles in the configuration is empty. Moreover, in order to show that this assembly is indeed a valid bit-reading gadget we show that in the presence of the bit writer tiles, only one of two bit reading assemblies (representing either a 0 or a 1) can assemble depending on the bit writer tiles.

To prove that each bit-reading gadget configuration can be used to obtain a valid assembly, we must compute the distances from the center of a given polygon to the center of another polygon. For convenience, we assume that the length of the apothem (the line segment from the center of a polygon to the midpoint of one of its sides) of all of the regular polygons is $\frac{1}{2}$, so that the distance from the centers of abutting polygons is 1. Then, let t be a polygonal tile, and let t' be a polygonal tile that abuts t. We say that a polygonal tile

has the *standard orientation* if after being translated so that it is centered at the origin, it has a side that corresponds to a vertical line l segment with midpoint at $\left(\frac{1}{2}, 0\right)$. See Figure 4.5a for a depiction of a polygonal tile with standard orientation that is also centered at the origin. For a polygonal tile with an odd number of sides, we say that a polygonal tile has *negated orientation* if after being translated so that it is centered at $(0, 0)$, it is the reflection of a tile which has standard orientation across the imaginary axis. This is depicted in Figure 4.5b.



(a) A polygonal tile with *standard orientation* with center $(0, 0)$.

(b) A polygonal tile with *negated orientation* with center $(0, 0)$.

Figure 4.5: Regular polygonal tile orientations

We enumerate the sides of t counter-clockwise starting from the side s_0 corresponding to l and ending at s_{n-1} where n is the number of sides of t. Similarly, if t has negated orientation, then we enumerate the sides as $\{s'_i\}_{i=0}^{n-1}$ as shown in Figure 4.5b. Then, relative to t, if t' abuts t along s_0, then the center of t' is $(1, 0)$. In general, for $\theta = \frac{2\pi}{n}$, if t' abuts t along s_m, then the center of t' is $(\cos(m\theta), \sin(m\theta))$. For the calculations in the following sections, it is convenient to identify $\mathbb{R}^2$ with the complex plane $\mathbb{C}$ so that (x, y) is identified with $x + iy$. Then according to Euler's formula, $(\cos(m\theta), \sin(m\theta)) \in \mathbb{R}^2$ corresponds to the complex number $e^{mi\theta} = \cos(m\theta) + i\sin(m\theta)$. In other words, when t has standard orientation, the centers of abutting polygons correspond to complex n^{th} roots of unity, as the centers correspond to the roots of the complex polynomial $x^n - 1 = 0$ (recall that n is the number of sides of t). Now let $\omega = e^{i\theta}$. Then these roots of unity are $\{\omega^i\}_{i=0}^{n-1}$. See Figure 4.7 for an example in the heptagonal tile case. Finally, notice that if t has negated

orientation and t' abuts t along s'_m, then the center of t' is $(-\cos(m\theta), -\sin(m\theta))$, and so the center of t' corresponds to $-\omega^m$.

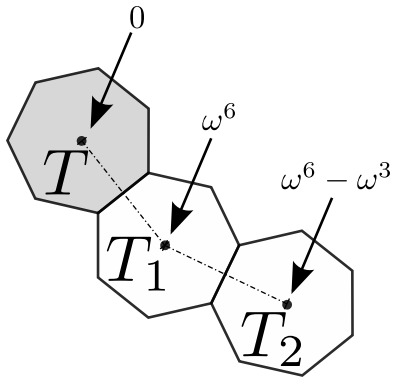


Figure 4.6: Relative to T, the center of T_1 corresponds to ω^6 and the center of T_2 corresponds to $\omega^6 - \omega^3$.

Now let $\mathcal{T}$ be a TAS with tiles of a single regular polygon shape, and let α be an assembly in $\mathcal{T}$ such that α contains a tile, t, with standard orientation and let t' be any tile in α (including t). Then, since addition (respectively, subtraction) of complex numbers corresponds to vector addition (respectively, subtraction) in $\mathbb{R}^2$, the center of t' corresponds to some polynomial in ω with integer coefficients. See Figure 4.6 for an example of the correspondence to the centers of heptagonal tiles to such polynomials.

4.4.1 Complex roots of unity example using heptagonal tiles

In this section, we give example assemblies using heptagonal tiles by computing the distances of relevant tile centers using 7^{th} roots of unity. Let $\omega = e^{\frac{2\pi}{7}}$. For a polygonal tile t with standard orientation, Figure 4.7(a) depicts the complex roots of unity corresponding to the centers of adjacent tiles. Similarly, for a polygonal tile t with negated orientation, Figure 4.7(b) depicts the negated complex roots of unity corresponding to the centers of adjacent tiles.

For a more in depth example of computing the centers of heptagonal tiles, consider the following TAS. Let $\mathcal{T}$ be the polygonal tile assembly system consisting of 10 tile types all with shape of a single regular heptagon. Moreover, suppose that each tile type has two

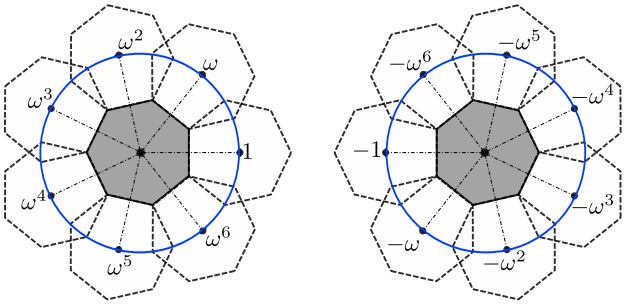


Figure 4.7: Representing the vector from the center of a heptagon (gray) to each center of an adjacent heptagon using the 7^{th} roots of unity.

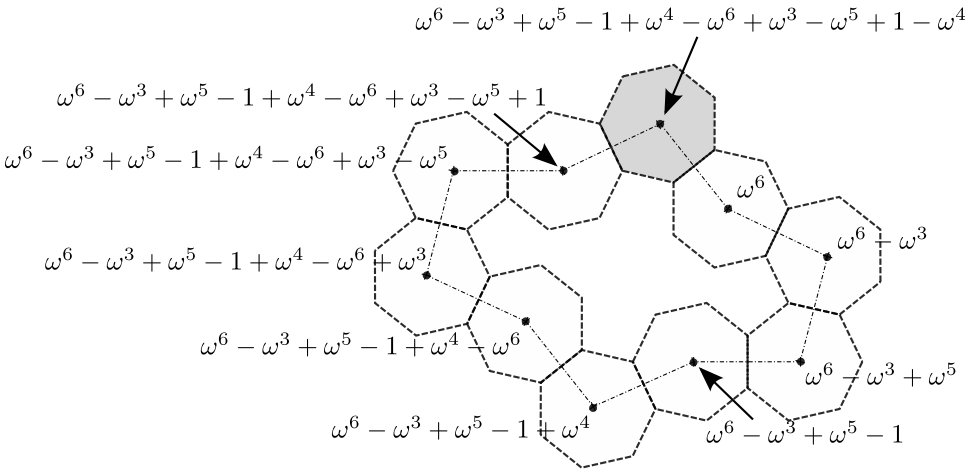
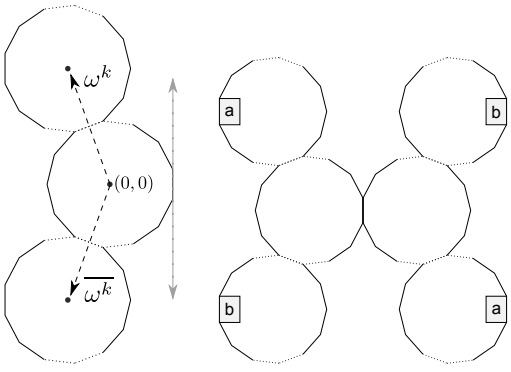


Figure 4.8: An example of computing the centers of heptagons using polynomials of complex roots of unity. The center of each heptagonal tile is labeled with a corresponding polynomial in ω. Glue labels are not shown.

edges with strength-1 glues, and that there are 10 glues appropriately defined so that starting from a single seed tile (the gray tile in Figure 4.8), the assemble proceeds until the closed "loop" of heptagonal tiles shown in Figure 4.8 assembles. At this point the assembly is terminal. Call this assembly α. Then let t be the seed tile. Keeping Figure 4.7 in mind, we can compute the centers of each polygonal tile in α relative to t. These are shown in Figure 4.8. In fact, we can even compute that the center of t to obtain the polynomial $\omega^6 - \omega^3 + \omega^5 - 1 + \omega^4 - \omega^6 + \omega^3 - \omega^5 + 1 - \omega^4$, and note that this polynomial is 0 reflecting the fact that α is a closed "loop" of heptagonal tiles.

4.5 Overview of polygonal grid construction

Given a regular polygon P, a *junction polyform* $\mathcal{P}$ is constructed in the following manner. We begin with a polygon in standard position centered at the origin. Starting from side s_0, we traverse the sides of the polygon counterclockwise until we come across the edge s_k where k is such that $\text{Re}(\omega^k) \leq 0$ and $j \geq k$ for all $j \in \mathbb{Z}$ such that $\text{Re}(\omega^j) \leq 0$. We place our next polygons of type P in non-standard positions centered at locations ω^k and $\overline{\omega^k}$ as shown in Figure 4.9a. Call this shape X. We create a new shape X' by reflecting X across the line $x = \frac{1}{2}$. We then take the union of the shapes X and X' obtaining our junction polyform shown in Figure 4.9b.



(a) Left half (b) Fully formed

Figure 4.9: Constructing a junction polyform.

We form a "grid" of these junction polyforms by attaching an infinite number of them

to each other so that the polygons with sides labeled "a" are adjacent to each other and the polygons labeled "b" are adjacent to each other. An example "grid" of these junction polyforms is shown in Figure 4.10.

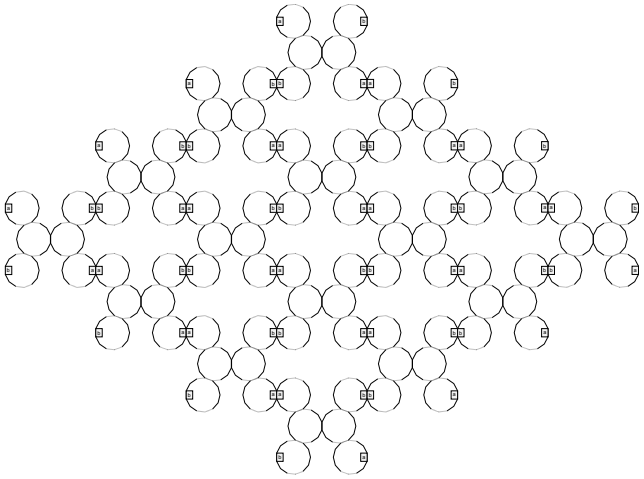


Figure 4.10: An example (where n=13) assembly formed by the system described in Section 4.5.

4.6 Polygonal grid construction

Given a polygon P, we now show how to form a lattice consisting of P. This grid will act as a coordinate system for our polygonal TAM systems and allow us to string several bit reading gadgets together so that we may simulate any Turing machine on any input. In order to do this, we first show that we can construct a single polyform from P which can "grid" the plane. It will then follow that we can form a lattice in the plane with P by placing polygons at the same locations and with same orientations as the polygons composing the grid formed with polyforms.

We begin by describing the construction of the polyform which we will use to construct our grid. We then show that this is indeed a valid polyform. Next, we shown that there exists a polygonal system which can tile the grid formed by the polyform.

Before we begin our construction, it is necessary to introduce a couple of definitions.

Definition. Let P be a regular polygon. A polyform $\mathcal{P}$ is a connected shape in the plane

which is constructed by combining a finite number of copies of P so that the following requirements are met:

1. the interior points of all instances of P are disjoint

2. every instance of P completely shares a common edge with some other instance of P.

The *bounding rectangle* B around a polyform $\mathcal{P}$ is the rectangle with minimal area that contains the interior points of $\mathcal{P}$.

Junction polyforms

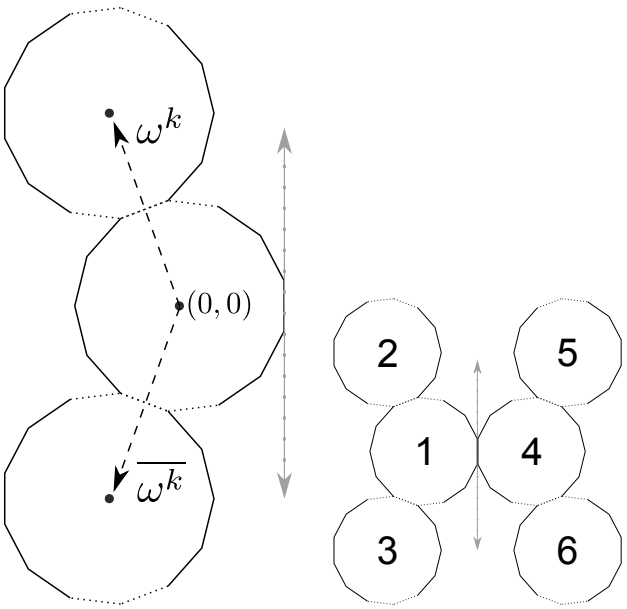
Given a regular polygon P, a *junction polyform* $\mathcal{P}$ is constructed in the following manner. We begin with a polygon which has standard orientation centered at the origin. Starting from side s_0, we traverse the sides of the polygon counterclockwise until we come across the edge s_k where k is such that $\text{Re}(\omega^k) <= 0$ and $j \geq k$ for all $j \in \mathbb{Z}$ such that $\text{Re}(\omega^j) <= 0$. We place our next polygons of type P with negated orientations centered at locations ω^k and $\overline{\omega^k}$ as shown in Figure 4.11a. Call this shape X. We create a new shape X' by reflecting X across the line $x = \frac{1}{2}$. We then take the union of the shapes X and X' obtaining our junction polyform shown in Figure 4.11b. We call k the polyform constant.

We now prove that this is indeed a valid polyform. First, we begin with some observations.

Observation 3. For any $n \in \mathbb{N}$ with $n > 2$, there exists a point p in the n^{th} roots of unity such that $-\frac{\sqrt{3}}{2} \leq \text{Re}(p) \leq 0$.

For $3 < n < 8$, this observation is mechanical. If $n >= 8$, the observation must hold since the n^{th} roots of unity are evenly spaced around the unit circle.

Observation 4. Let P be a regular polygon with n sides in standard orientation. Also, let $k \in \mathbb{N} \cup \{0\}$ be such that $k \leq n$ and $\text{Im}(-\omega^k) \leq 0$. Denote the vertices that compose side s_k by $\vec{v_l}$ and $\vec{v_r}$ where $\vec{v_l}$ is the counterclockwise most vertex and $\vec{v_r}$ is the clockwise most vertex. Set $\vec{v} = \vec{v_r} - \vec{v_l}$. Then the following hold:



(a) The left half of a junction polyform.

(b) The fully formed junction polyform.

Figure 4.11: The construction of a junction polyform.

1. if $\text{Re}(-\omega^k) > 0$, then $\text{Im}(\vec{v}) > 0$, and

2. if $\text{Re}(-\omega^k) \leq 0$, then $\text{Im}(\vec{v}) \leq 0$.

This observation falls out of the fact that ω^k and $\vec{v}$ must be orthogonal.

Observation 5. Let k be the polyform constant for some polyform composed of regular polygons with n sides. Let P be a regular polygon with n sides centered at the origin in standard orientation. Then

1. the clockwise most vertex that composes s'_k is a southernmost point in P, and

2. the location of the counterclockwise most vertex that composes s_k, call this point $\vec{z}$, is such that $\text{Im}(\vec{z}) \geq 0$.

To see the first part of this observation, note that $\text{Re}(-\omega^{k-1}) < 0$. This along with the observation 4 implies that the clockwise most vertex of side s'_{k-1} must lie to the north of the clockwise most vertex that composes s'_k. Note that the clockwise most vertex of side s'_{k+1} also must not lie to the south of the counterclockwise most vertex of side s'_k. Consequently,

because P is convex, the clockwise most vertex that composes s'_k is a southernmost point in P.

The second part of this observation follows from Observation 3 and the fact that a regular polygon in standard orientation centered at the origin will always have a vertex with an absent imaginary part and a real part that is less than 0.

Lemma 10. Let P be a regular polygon, and let k be the junction polyform constant obtained from the junction polyform composed of P. Then the sets of interior points of the following polygons are pairwise disjoint: 1) the polygon in standard orientation centered at the origin, 2) the polygon with negated orientation centered at ω^k, and 3) the polygon with negated orientation centered at $\overline{\omega^k}$.

Proof. It follows from the discussion in Section 4.4 that the interior points of the polygon centered at the origin and the polygon centered at ω^k are disjoint. Also since the complex conjugate of a root of unity is also a root of unity, it follows from the discussion in Section 4.4 that the interior points of the polygon centered at the origin and the polygon centered at $\overline{\omega^k}$ are disjoint.

It is left to show that the interior points of the two polygons centered at the roots of unity are disjoint. To see this, first note that it follows from Observation 5 that no interior point of the polygon centered at the location ω^k has real part that is less than or equal to 0. Indeed, first note that the clockwise most vertex of side s'_k of the polygon centered at location ω^k will overlap the counterclockwise most vertex of side s_k of the polygon centered at the origin by construction. It follows immediately from Observation 5 that all interior points in the polygon centered at ω^k have imaginary parts greater than 0. Since the polygon centered at $\overline{\omega^k}$ is a reflected copy of the polygon centered at $\overline{\omega^k}$, it follows that the interior points in this polygon have imaginary parts less than 0. Consequently, the interior points of the two polygons are disjoint.

□

Lemma 11. Given a regular polygon P, the junction polyform constructed above is indeed

a valid polyform.

Proof. To see that the junction polyform constructed above is a valid polyform, we check that all of the requirements in the definition of polyform are met. Since the center of polygons labeled "2" and "3" are each located at one of the n^{th} roots of unity, it follows from the discussion in Section 4.4 that polygons labeled "1" and "2" as well as polygons labeled "1" and "3" are joined along a common edge and share that edge entirely. This same line of reasoning shows that the polygon labeled "4" is joined along a common edge and shares that edge entirely with the polygon labeled "1". Since the shape formed by polygons labeled "4", "5" and "6" is a reflection of the left side of the shape, all of the polygons are joined along a common edge and shares that edge entirely. It is readily seen from this argument that our shape is also connected.

It is now left to show that no two polygons in the shape overlap. We denote the polyform constant obtained from P by k. It follows from Lemma 10 that the interior points of the polygons labeled "1", "2", and "3" are pairwise disjoint. Since, the polygons labeled "4", "5", and "6" are a reflection of the polygons labeled "1", "2", and "3", they too are pairwise disjoint. To show that the polygons in the two reflected halves of the shape are pairwise disjoint, first observe that the centers of the polygons labeled "2" and "3" have real parts less than or equal to the real part of the polygon labeled "1". Consequently, after the reflection and "attachment" of the two halves, the polygons labeled "2" and "5" and the polygons labeled "3" and "6" have no less distance between each other than the polygons labeled "1" and "4". Since the polygons labeled "1" and "4" have disjoint interior points, it follows that the polygons mentioned above have disjoint interior points. Consequently, no two polygons in the shape overlap.

□

Polygonal Grid Technical Lemmas

The following lemma will assist us in proving Lemma 13. Informally, it states that the bounding rectangle of the junction polyform described above and shown in Figure 4.11b will "touch" sides s_0' of the polygons labeled "2" and "3" and sides s_0 of the polygons labeled "5" and "6". This will imply that we can attach the polyform junctions by attaching sides s_0 of polygons labeled "5" and "6" to sides s_0' of polygons labeled "5" and "6".

Lemma 12. Consider the polygons composing the junction polyform $\mathcal{P}$ constructed above from some regular polygon P (shown in Figure 4.11b). Also, let B be the bounding rectangle around $\mathcal{P}$. Let E be the set of points consisting of the union of the following sets of points: 1) the set of boundary points on side s_0' of the polygon labeled "2", 2) the set of boundary points on side s_0' of the polygon labeled "3", 3) set of boundary points on side s_0 of the polygon labeled "5", and 4) the set of boundary points on side s_0 of the polygon labeled "6". Then $E \subset E \cap B$.

Proof. We prove that the boundary points on side s_0 of the polygons labeled "4" and "6" in Figure 4.11b lie on the bounding rectangle B. The proof that the boundary points on side s_0' of the polygons labeled "2" and "3" lie on the bounding rectangle will then follow from a similar argument.

First, observe that for a polygon P with standard position centered at the origin, the boundary points on side s_0 are the easternmost points contained in the polygon. Furthermore, all of these points lie on the line $x = \frac{1}{2}$. Now note that by our construction of the junction polyform, one of the tiles labeled "5" and "6" will contain the easternmost point of the polyform. Indeed, let x_4 be the real part of the point in the center of the polygon labeled "4". Since our construction ensures the real part of the point in the center of the polygon labeled "5" is of the form $x_4 + r$ for $r \in [0, \frac{\sqrt{3}}{2}]$, the polygon labeled "5" will contain a point as east or further east than the points in the polygon labeled "4".

We claim that the polygons labeled "5" and "6" have centers with equal real parts. To see this, recall that the centers of the polygons labeled "2" and "3"have the same real

parts since they are conjugates of each other. Since the polygons labeled "5" and "6" are in the same position relative to each other as the polygons labeled "2" and "3" just reflected across the line $y = \frac{1}{2}i$, it follows that the polygons labeled "5" and "6" have equal real parts.

From our construction of the junction polyform, it is clear that none of the polygons labeled "1", "2", or "3" have a point that is an easternmost point of the polyform. Thus, the s_0 sides of the polygons labeled "5" and "6" are all easternmost points of the polyform. Consequently, these points lie on the bounding box B. □

Observation 6. Let P be a regular polygon, $\mathcal{P}$ be a polyform junction formed from P, B be the bounding rectangle for $\mathcal{P}$, and let k be the polyform constant. Furthermore, let h_b be the height of the bounding rectangle and let h_w be the width of the bounding rectangle. Then, the following constraints hold for h_b and h_w: 1) $h_b \leq 4\,\mathrm{Im}(\omega^k)$ and 2) $h_w = 2\,\mathrm{Re}(-\omega^k + 1)$.

Figure 4.12 shows the dimensions of the polyform. Note that the width of the polyform is clearly $2\,\mathrm{Re}(-\omega^k + 1)$. To see that $h_b \leq 4\,\mathrm{Im}(\omega^k)$, note that by the way we constructed the junction polyform no interior points of the polyform can lie on the dotted lines shown in Figure 4.12. Since the distance between these two dotted lines is $4\,\mathrm{Im}(\omega^k)$, it must be the case that $h_b \leq 4\,\mathrm{Im}(\omega^k)$.

The next lemma states that given any regular polygon, we can form a a periodic grid of the plane.

Constructing the polygonal grid

Lemma 13. Given a regular polygon P, there exists a directed, polygonal tile system $\mathcal{T} = (T, \sigma)$ (where the seed is centered at location $(0, 0)$ and the tile set T contains a tile t) and vectors $\vec{v}, \vec{w} \in \mathbb{Z}^2$, such that $\mathcal{T}$ produces the terminal assembly α, which we refer to as a *grid*, with the following properties. (1) Every position in α of the form $c_1\vec{v} + c_2\vec{w}$, where $c_1, c_2 \in \mathbb{Z}$, is occupied by the tile t, and (2) for every $c_1, c_2 \in \mathbb{Z}$, the position in $\mathbb{Z}^2$ of the

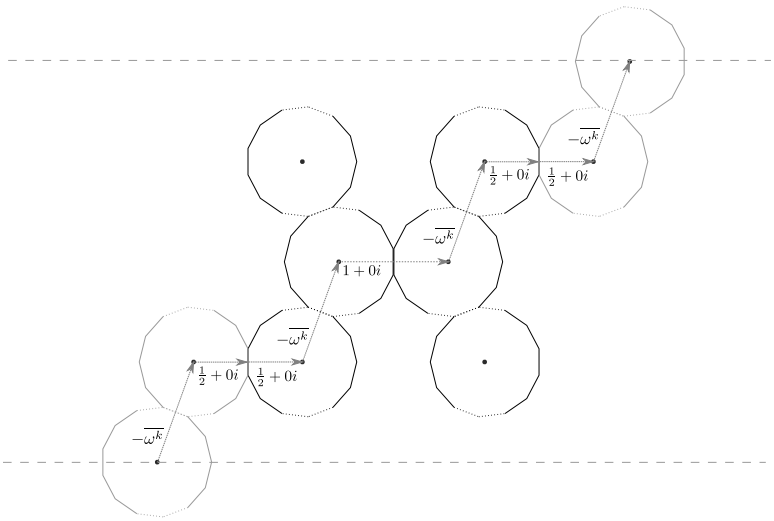


Figure 4.12: The vectors showing the dimensions of the polyforms.

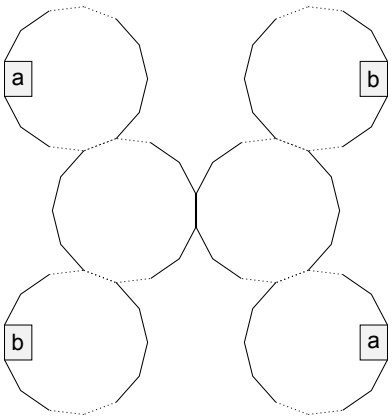


Figure 4.13: The preformed assembly which is composed of the tile set of the system described in the proof of Lemma 13. The preformed assembly has two glues labeled "a" and "b" placed as shown.

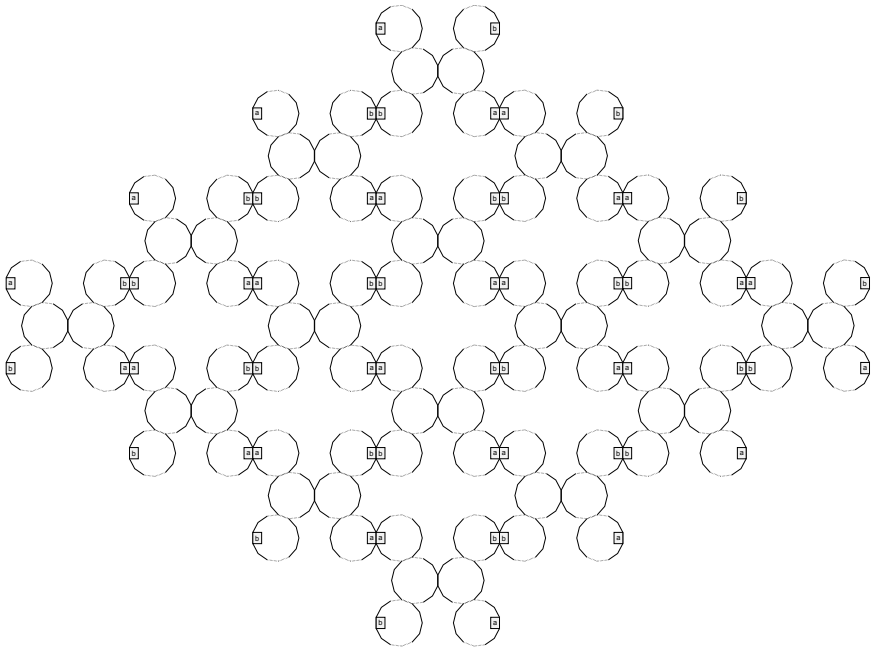
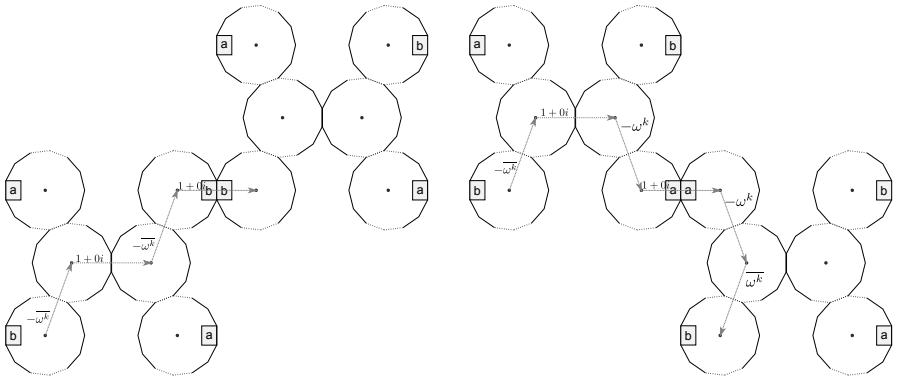


Figure 4.14: An assembly formed by the system described in the proof of Lemma 13.



(a) The path of vectors which yields the vector $\vec{v}$.

(b) The path of vectors which yields the vector $\vec{w}$.

Figure 4.15: Choosing the vectors $\vec{v}$ and $\vec{w}$.

form $c_1\vec{v} + c_2\vec{w}$ is occupied by the tile t.

Proof. For the first part of this proof, we think of our polygonal tile system as first forming the junction polyform $\mathcal{P}$ before attaching it to our assembly. Later on in the proof, we will see that this is a valid assumption. Our tile set T, will consist of tiles of shape P that form the junction polyform with the glues labeled "a" and "b" exposed as shown in Figure 4.13. Note that for the first part of the proof we are essentially thinking of the assembly shown in Figure 4.13 as a tile. Thus, we refer to the junction polyform as a tile and we refer to a polygon composing the polyform as a pixel. More formally, a pixel in the polyform is a location in the complex plane given by the center of a tile in the polyform shown in Figure 4.11 (where we assume that the center of the tile labeled "1" is placed at the origin).

To begin, we position our single seed so that the polygon labeled "1" in Figure 4.11b is centered at the origin. An assembly formed by such a system is shown in Figure 4.14.

Let $\mathcal{P}$ be a junction polyform composed of the polygon P and let k be the polyform constant as discussed in the construction of the junction polyform. Set $\vec{v} = -\overline{\omega^k} + (1+0i) - -\overline{\omega^k} + (1+0i) = -2\overline{\omega^k} + 2(1+0i)$ and $\vec{w} = -\overline{\omega^k} + (1+0i) - \omega^k + (1+0i) - \omega^k + \overline{\omega^k} = -2\omega^k + 2(1+0i)$. The intuition behind choosing these vectors is shown in Figure 4.15a and Figure 4.15b.

The following terminology is borrowed from [13]. Define $\mathcal{P}[i,j] = p + i \cdot \vec{v} + j \cdot \vec{w}$ for $i, j \in \mathbb{Z}^2$. Here, p acts as a distinguished pixel that we use as a reference point. Then, for two polyforms $\mathcal{P}[i,j]$ and $\mathcal{P}[k,l]$, we say that these polyforms are neighboring if $i = k$ and $|j - l| = 1$ or $j = l$ and $|i - k| = 1$.

As in [13] we prove the following claim.

Claim: $\mathcal{P}[i,j]$ for all $i, j \in \mathbb{Z}^2$ defines a grid of non-overlapping polyforms such that any two neighboring polyforms $\mathcal{P}[i,j]$ and $\mathcal{P}[k,l]$ contain pixels with a shared edge. Such a grid of polyforms is shown in Figure 4.14.

To begin, we show that if $i \neq k$ or $j \neq l$, then the interior points of $\mathcal{P}[i,j]$ and $\mathcal{P}[k,l]$ are disjoint. Let $a = (k - i)$ and $b = (l - j)$. In order to show that $\mathcal{P}[i,j]$ does not overlap

$\mathcal{P}[k,l]$, we show that 1)$|\operatorname{Re}(a\vec{v}+b\vec{w})| \geq |2\operatorname{Re}(-\omega^k+1)|$ or 2) $|\operatorname{Im}(a\vec{v}+b\vec{w})| \geq |4\operatorname{Im}(\omega^k)|$. Since, by Lemma 6, these are the dimensions of the bounding box of $\mathcal{P}$, it will then follow that their interiors are disjoint.

We consider three cases 1) $a+b>0$, 2) $a+b=0$, and 3) $a+b<0$. First note that

$$\begin{aligned} a\vec{v}+b\vec{w} &= a(-2\overline{\omega^k}+2(1+0i))+b(-2\omega^k+2(1+0i)) \\ &= -2(a\overline{\omega^k}+b\omega^k)+2(a+b) \end{aligned}$$

For case 1, observe that

$$\begin{aligned} |\operatorname{Re}(a\vec{v}+b\vec{w})| &= |\operatorname{Re}(-2(a\overline{\omega^k}+b\omega^k)+2(a+b))| \\ &= |-2(a\operatorname{Re}(\overline{\omega^k})+b\operatorname{Re}(\omega^k))+2(a+b)| \\ &= |-2\operatorname{Re}(\omega^k)(a+b)+2(a+b)| \\ &\geq |-2\operatorname{Re}(\omega^k)+2|. \end{aligned}$$

In the case that $a+b=0$, we have

$$\begin{aligned} |\operatorname{Im}(a\vec{v}+b\vec{w})| &= |\operatorname{Im}(-2(a\overline{\omega^k}+b\omega^k)+2(a+b))| \\ &= |\operatorname{Im}(-2(a\overline{\omega^k}+b\omega^k))| \\ &= |\operatorname{Im}(-2((-b)\overline{\omega^k}+b\omega^k))| \\ &= |-2(b)(\operatorname{Im}(-\overline{\omega^k})+\operatorname{Im}(\omega^k))| \\ &= |-2(b)(2\operatorname{Im}(\omega^k))| \\ &\geq |-4\operatorname{Im}(\omega^k)|. \end{aligned}$$

Although case 3 is similar to case 1, we include it here for completeness. If $a+b<0$,

notice that

$$\begin{aligned} |\operatorname{Re}(a\vec{v}+b\vec{w})| &= |\operatorname{Re}(-2(a\overline{\omega^k}+b\omega^k)+2(a+b))| \\ &= |-2(a\operatorname{Re}(\overline{\omega^k})+b\operatorname{Re}(\omega^k))+2(a+b)| \\ &= |-2\operatorname{Re}(\omega^k)(a+b)+2(a+b)| \\ &\geq |2\operatorname{Re}(\omega^k)-2|. \end{aligned}$$

Now suppose that $\mathcal{P}[i,j]$ and $\mathcal{P}[k,l]$ are neighboring polyforms. First, suppose that $i = k$ and $|j-l| = 1$. We consider the case where $l = j+1$ and note that the case where $l = j-1$ is similar. Consider the polygons in the lower left hand corner of the bounding rectangle of the polyforms and denote this polygon p. Note that the polygon p in $\mathcal{P}[k,l]$ lies at a position

$$\begin{aligned} (k\vec{v}+l\vec{w})-(i\vec{v}+j\vec{w}) &= (i\vec{v}+(j+1)\vec{w})-(i\vec{v}+j\vec{w}) \\ &= \vec{w} \end{aligned}$$

relative to the polygon p in $\mathcal{P}[i,j]$.

Now, notice that $\mathcal{P}[i,j]$ has a polygon that lies at position $-\overline{\omega^k}+(1+0i)-\omega^k$ relative to p in $\mathcal{P}[i,j]$(this is the polygon that lies in the bottom right hand corner of the bounding box), and $\mathcal{P}[k,l]$ has a polygon that lies at position $-\overline{\omega^k}+\omega^k$ relative to p in $\mathcal{P}[k,l]$ (this is the polygon that lies in the top left hand corner of the bounding box). Call the first pixel described p' and the latter p''. Observe that by the construction of the junction polyform, p' has standard orientation and p'' has negated orientation. Furthermore, observe that p'' lies at location

$$\begin{aligned} &(\vec{w}+(-\overline{\omega^k}+\omega^k)-(-\overline{\omega^k}+(1+0i)-\omega^k) \\ &= -2\omega^k+2(1+0i)+(-\overline{\omega^k}+\omega^k)-(-\overline{\omega^k}+(1+0i)-\omega^k) \\ &= (1+0i) \end{aligned}$$

relative to p'. Since p' has standard orientation, p'' has negated orientation and p'' lies at position $(1 + 0i)$ relative to p', it follows from the discussion in Section 4.4 that polygon p' and polygon p'' completely share a common edge.

Conversely, assume that $j = l$ and $|i - k| = 1$. We consider the case where $k = i - 1$, and, once again, note that the case where $k = i + 1$ is similar. Notice that the polygon p in $\mathcal{P}[k, l]$ lies at a position

$$\begin{aligned}(k\vec{v} + l\vec{w}) - (i\vec{v} + j\vec{w}) &= ((i-1)\vec{v} + j\vec{w}) - (i\vec{v} + j\vec{w}) \\ &= -\vec{v}\end{aligned}$$

relative to the polygon p in $\mathcal{P}[i, j]$.

Denote the polygon that lies at position $-2\overline{\omega^k} + (1 + 0i)$ relative to p in $\mathcal{P}[k, l]$ by p' (this is the polygon that lies in the top right hand corner of the bounding box). Observe that, relative to polygon p in $\mathcal{P}[i, j]$, the polygon p' in $\mathcal{P}[k, l]$ lies at position

$$\begin{aligned}-\vec{v} + (-2\overline{\omega^k} + (1 + 0i)) &= -(-2\overline{\omega^k} + 2(1 + 0i)) + (-2\overline{\omega^k} + (1 + 0i)) \\ &= -(1 + 0i).\end{aligned}$$

Since p in $\mathcal{P}[i, j]$ has negated orientation, p' in $\mathcal{P}[k, l]$ has standard orientation, and p' lies at a position $-(1 + 0i)$ relative to p, it follows from the discussion in Section 4.4 that polygon p and polygon p'' completely share a common edge.

Now, note that since none of the "polyform junction tiles" overlap, there are not any race conditions. Consequently, we can build the assembly described above by attaching one polygon tile at a time (instead of an assembly of polygons). The seed of our assembly will be the southwest tile of $\mathcal{P}[0, 0]$. □

4.6.1 Grid notation

For some polygon P, we let g_α denote the terminal assembly of the tile system given in Lemma 13 (i.e. the grid assembly obtained from P). Furthermore, for a tile system $\mathcal{T}$ of shape P, $\alpha \in \mathcal{A}[\mathcal{T}]$, and t a tile of α centered at the location $\vec{x}$, we say that t is *on grid* with respect to g_α if there exists a tile $t' \in g_\alpha$ such that t' is centered at the location $\vec{x}$ and has the same orientation of t. If there does not exist such a t', then we say that t is *off grid* with respect to g_α.

4.6.2 *Normalized* bit-reading gadgets

Let a bit reading gadget have the properties that: 1)the tile from which the bit writer begins growth is on grid, 2) the last tile to be placed in the bit writer is on grid, and 3) the tile t from which the bit reader grows is also placed on grid. We call such a bit-reading gadget an *on grid bit-reading gadget*. A pair of *normalized bit-writers* α_{u0} and α_{u1} have the property that 1) α_{u0} and α_{u1} are the two bit writers for some bit reading gadget and 2) the location and position of the first tile placed in the two assemblies is the same as well as the location and position of the last tile placed. A *normalized bit-reading gadget* is an on grid bit-reading gadget with normalized bit-writers.

4.7 Polygons which "can not compute" at temperature 1

In this section, we prove Theorem 6 by showing a set of polygons for which it is impossible to create bit-reading gadgets at $\tau = 1$, namely regular polygons with less than 7 sides (i.e. equilateral triangles, regular pentagons, and regular hexagons), as this was already shown to be true for squares in [13]. This provides a sharp dividing line, since we have shown that all regular polygons with ≥ 7 sides can form bit reading gadgets, and thus are capable of universal computation, at $\tau = 1$.

We now restate the Theorem for completeness and give its proof.

Theorem 7. Let $n \in \mathbb{N}$ be such that $3 \leq n \leq 6$. Then, there exists no temperature 1 single-shaped polygonal tile assembly system $\mathcal{T} = (T, \sigma, 1)$ where for all $t \in T$, t is a regular polygon with n sides, and a bit-reading gadget exists for $\mathcal{T}$.

To prove Theorem 7, we break it into two main cases and prove lemmas about (1) equilateral triangles and hexagons, and (2) pentagons.

4.7.1 Equilateral triangles, squares, and regular hexagons

Equilateral triangles, squares, and regular hexagons are all capable of tessellations of the plane. That is, using tiles of only one of those shapes it is possible to tile the entire plane with no gaps. (As a side note, these are the only regular polygons which can do so.) In a system consisting of tiles of only one of those shapes, all tiles must be placed into positions aligning with a regular grid (i.e. no tile can be offset or rotated from the grid). It was shown in [13] that squares cannot form bit-reading gadgets at $\tau = 1$, and because of the tessellation ability of equilateral triangles and regular hexagons and their restriction to fixed grids, the proof of [13] can be extended in a straightforward way to also prove that equilateral triangles and regular hexagons cannot form bit reading gadgets at $\tau = 1$. Thus, the following proof is nearly identical to that for squares of [13].

Lemma 14. There exists no temperature 1 polygonal tile assembly system $\mathcal{T} = (T, \sigma, 1)$ where for all $t \in T$, t is an equilateral triangle, and a bit-reading gadget exists for $\mathcal{T}$.

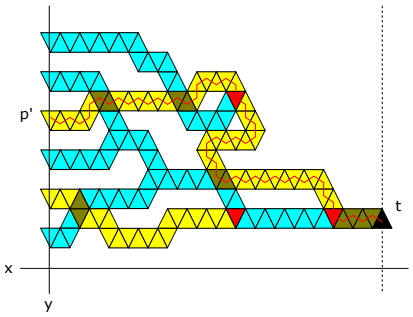
Lemma 15. There exists no temperature 1 polygonal tile assembly system $\mathcal{T} = (T, \sigma, 1)$ where for all $t \in T$, t is a regular hexagon, and a bit-reading gadget exists for $\mathcal{T}$.

Proof. We prove Lemmas 14 and 15 by contradiction. Also, since each will use exactly the same arguments, we will prove both simultaneously and note the single location in the proof where the shapes of the tiles is relevant. Therefore, assume that there exists a single-shape system $\mathcal{T} = (T, \sigma, 1)$ such that $\mathcal{T}$ has a bit-reading gadget. (Without loss of generality, assume that the bit-reading gadget reads from right to left and has the same

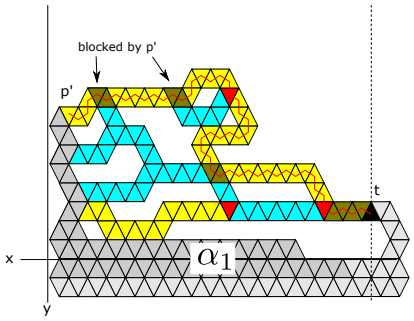
orientation as in Definition 4.3.2.) Let (t_x, t_y) be the coordinate of the tile t from which the bit-reading paths originate (recall that it is the same coordinate regardless of whether or not a 0 or a 1 is to be read from α_0 or α_1, respectively). By Definition 4.3.2, it must be the case that if α_0 is the only portion of α in the first quadrant to the left of t, then at least one path can grow from t to eventually place a tile from T_0 at $x = 0$ (without placing a tile below $y = 0$ or to the right of $(t_x - 1)$). We will define the set P_0 as the set of all such paths which can possibly grow. Analogously, we will define the set of paths, P_1, as those which can grow in the presence of α_1 and place a tile of a type in T_1 at $x = 0$. Note that by Definition 4.3.2, neither P_0 nor P_1 can be empty.

Since all paths in P_0 and P_1 begin growth from t at (t_x, t_y) and must always be to the left of t, at least the first tile of each must be placed in location $(t_x - 1, t_y)$. We now consider a system where t is placed at (t_x, t_y) and is the only tile in the plane (i.e. neither α_0 nor α_1 exist to potentially block paths), and will inspect all paths in P_0 and P_1 in parallel. If all paths follow exactly the same sequence of locations (i.e. they overlap completely) all the way to the first location where they place a tile at $x = 0$, we will select one that places a tile from T_0 as its first at $x = 0$ and call this path p_0, and one which places a tile from T_1 as its first at $x = 0$ and call it p_1. This situation will then be handled in Case (1) below. In the case where all paths do not occupy the exact same locations, then there must be one or more locations where paths branch. Since all paths begin from the same location, we move along them from t in parallel, one tile at a time, until the first location where some path, or subset of paths, diverge. At this point, we continue following only the path(s) which take the clockwise-most branch. We continue in this manner, taking only clockwise-most branches and discarding other paths, until reaching the location of the first tile at $x = 0$. (Figures 4.16a and 4.17a show examples of this process.) We now check to see which type(s) of tiles can be placed there, based on the path(s) which we are still following. We again note that by Definition 4.3.2, some path must make it this far, and must place a tile of a type either in T_0 or T_1 there. If there is more than one path remaining,

since they have all followed exactly the same sequence of locations, we randomly select one and call it p'. If there is only one, call it p'. Without loss of generality, assume that p' can place a tile from T_0 at that location. This puts us in Case (2) below.

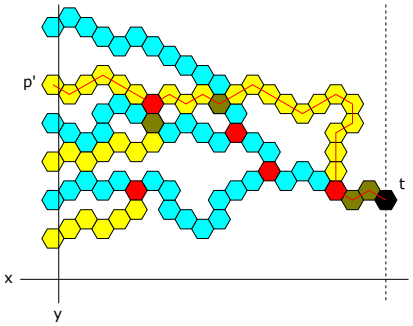


(a) Example sets P_0 and P_1, with p' traced with a red line. Red triangles represent branching points of paths, gold triangles represent overlapping points of different branches.

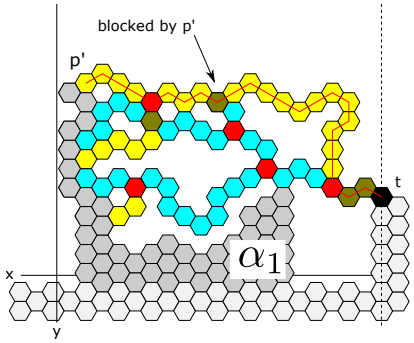


(b) An example of the growth of p' (traced with a red line) blocked by α_1. By first letting as much of p' grow as possible, it is guaranteed that all other paths must be blocked from reaching $x = 0$.

Figure 4.16: Failed bit-readers with equilateral triangles.



(a) Example sets P_0 and P_1, with p' traced with a red line. Red hexagons represent branching points of paths, gold hexagons represent overlapping points of different branches.



(b) An example of the growth of p' (traced with a red line) blocked by α_1. By first letting as much of p' grow as possible, it is guaranteed that all other paths must be blocked from reaching $x = 0$.

Figure 4.17: Failed bit-readers with regular hexagons.

Case (1) Paths p_0 and p_1 occupy the exact same locations through all tile positions and the placement of their first tiles at $x = 0$. Also, there are no other paths which can grow from t, so, since by Definition 4.3.2 some path must be able to complete growth in the

presence of α_0, either must be able to. Therefore, we place α_0 appropriately and select an assembly sequence in which p_1 grows, placing a tile from T_1 as its first at $x = 0$. This is a contradiction, and thus Case (1) cannot be true.

Case (2) We now consider the scenario where α_1 has been placed as the bit-writer according to Definition 4.3.2, and with t at (t_x, t_y). Note that path p' must now always, in any valid assembly sequence, be prevented from growing to $x = 0$ since it places a tile from T_0 at $x = 0$, while some path from P_1 must always succeed. We use the geometry of the paths of P_1 and path p' to analyze possible assembly sequences.

We create a (valid) assembly sequence which attempts to first grow only p' from t (i.e. it places no tiles from any other branch). If p' reaches $x = 0$, then this is not a valid bit-reader and thus a contradiction. Therefore, p' must not be able to reach $x = 0$, and since the only way to stop it is for some location along p' to be already occupied by a tile, then some tile of α_1 must occupy such a location. This means that we can extend our assembly sequence to include the placement of every tile along p' up to the first tile of p' occupied by α_1, and note that by the definition of the regular grid of equilateral triangle tiles, or of regular hexagon tiles, some tile of p' must now have a side adjacent to some tile of α_1. At this point, we can allow any paths from P_1 to attempt to grow. However, by our choice of p' as the "outermost" path due to always taking the clockwise-most branches, any path in P_1 (and also any other path in P_0 for that matter) must be surrounded in the plane by p', α_1, and the lines $y = 0$ and $x = t_x$ (which they are not allowed to grow beyond), and thus cannot be connected and extend beyond that boundary. (Examples can be seen in Figures 4.16b and 4.17b.) Therefore, no path from P_1 can grow to a location where $x = 0$ without colliding with a previously placed tile or violating the constraints of Definition 4.3.2. (This situation is analogous to a prematurely aborted computation which terminates in the middle of computational step.) This is a contradiction that this is a bit-reader, and thus none must exist. □ □

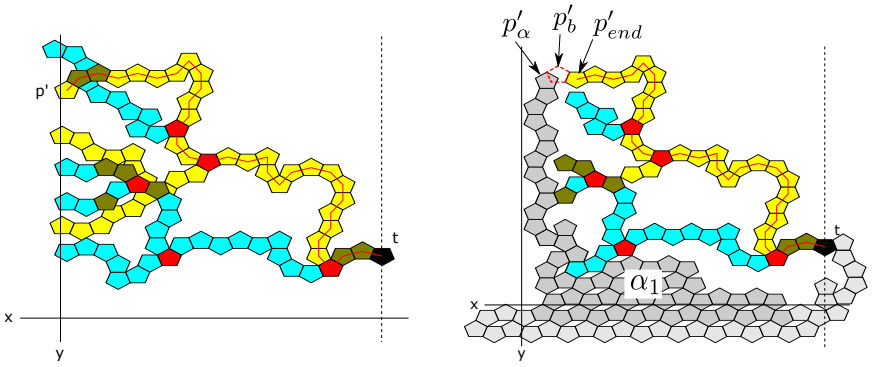
4.7.2 Regular pentagons

Because regular pentagons don't tessellate the plane, the proof that they can't form bit-reading gadgets is slightly different than for equilateral triangles, squares, and regular hexagons. However, the fact that they can only bind in two relative rotations and the ratio of their side lengths to perimeters ensure that they are still unable to form bit-reading gadgets due to the fact that it is still impossible for one path of regular pentagons to be blocked from continued growth without trapping all other paths on one side. This means that the "outermost" path, along with any part of the bit-writer which blocks its full growth, can always prevent any inner paths from sufficient growth.

Lemma 16. There exists no temperature 1 polygonal tile assembly system $\mathcal{T} = (T, \sigma, 1)$ where for all $t \in T$, t is a regular pentagon, and a bit-reading gadget exists for $\mathcal{T}$.

Proof. The proof of Lemma 16 is nearly identical to that for Lemmas 14 and 15, with the only slight change being due to the fact that regular pentagons aren't constrained to a single fixed grid. First, because of this we will slightly adapt Definition 4.3.2 so that rather than requiring tiles to be at specific discrete coordinates, they instead are constrained by lines in $\mathbb{R}^2$. For instance, we no longer require the bit-reader to grow a path to x-coordinate 0, but instead just beyond a set vertical line $x = r$ for some $r \in \mathbb{R}$. (without loss of generality we'll assume $x = r = 0$ for that constraint.) This change is merely a technicality and does not affect the proof, and therefore, we will use the previous proof up to the point where Case (2) makes the argument that the regular grid of tiles ensures that the last tile which can be placed along p' must have an edge adjacent to a tile of α_1. Due to the lack of such a grid, we will now only be able to guarantee that some portion of the next position of p', i.e. the location where α_1 first prevents the addition of another tile (which we will now refer to as location p'_b), is occupied by a tile of α_1 (whose location we will now refer to as p'_α. Referring to the location of the last tile which can be placed on p' as p'_{end}, by the fact that p' would have been a connected path which included p'_b, and that the tile at p_α prevents its placement, the location p'_b must consist of the area of a tile ori-

ented so that it has an edge adjacent to p'_{end}. Also, although the tiles at p'_α and p_{end} need not share an adjacent edge and there may in fact be a gap between them, p'_α must overlap with p'_b. (See Figure 4.18b for an example.)



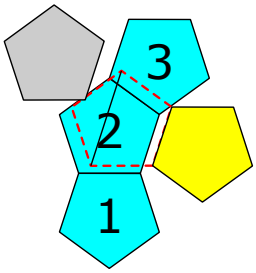
(a) Example sets P_0 and P_1, with p' traced with a red line. Red pentagons represent branching points of paths, gold hexagons represent overlapping points of different branches.

(b) An example of the growth of p' (traced with a red line) blocked by α_1. By first letting as much of p' grow as possible, it is guaranteed that all other paths must be blocked from reaching $x \leq 0$. The location outlined in dashed red represents the location of the first tile of p' which is blocked by α_1.

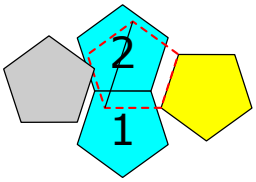
Figure 4.18: Failed bit-readers with regular pentagons.

At this point, we continue the direction of the previous proof and allow any paths from P_1 to attempt to grow. However, by our choice of p' as the "outermost" path due to always taking the clockwise-most branches, any path in P_1 must be surrounded in the plane by p', α_1, and the lines $y = 0$ and $x = t_x$ (which they are not allowed to grow beyond), with the only discontinuity being the possible gap consisting of the portion of p'_b which is not occupied by the tile at p'_α. We prove that this gap must be insufficient to allow a path p from P_1 to grow through using a simple case analysis. A key feature of regular pentagonal tiles is the fact that although their relative offsets are not fixed on a grid, their relative rotations are constrained to a total of only two orientations while allowing them to be connected to the same assembly.

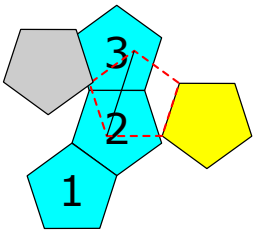
To prove the gap is insufficient for p, we perform a case analysis as outlined in Figure 4.19. We first note that the blocker may never occupy space inside the black diagonal



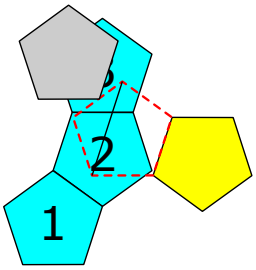
(a) Tile 2 has the same orientation as p'_b and attaches at an offset slightly below



(b) Tile 2 has the same orientation as p'_b and attaches at an offset slightly above



(c) Tile 2 has the opposite orientation as p'_b and attaches with its nearest vertex below the bottom corner of p'_{end} and p'_b



(d) Tile 2 has the opposite orientation as p'_b and attaches with its nearest vertex above the bottom corner of p'_{end} and p'_b

Figure 4.19: A case analysis of why one path, p', of regular pentagonal tiles cannot be blocked while allowing another, p, to grow through a gap. Let the yellow tile be at p'_{end}, the location of the last placed tile of path p', the grey represent some blocking tile of α_1, at p'_α (which may be in either possible orientation), and the red dashed location be p'_b, the first tile of p' prevented from being placed. The blue tiles represent a portion of path p which attempts to grow from below to above p', with the tiles labeled in order of their placements.

shown across a portion of the red dashed box, since that would leave a maximum distance of one side length for the gap throughout a portion of the gap, and for any pair of regular pentagonal tiles, the narrowest location is never less than that, occurring at the boundary of two adjacent tiles and immediately increasing on both sides of that. We now analyze the various cases.

In Figure 4.19a, having the tile at position 2 at the same orientation but an offset below p'_b requires that that tile fill the bottom edges of the location p'_b, leaving the blocker only the top left edge through which to block. However, in order to allow the tile at location 3 to bind to the top right side, the tile at location 2 must be offset up and left in order not to collide with the yellow tile at p'_{end} (since the width of the pair of adjacent tiles increases on both sides of their adjacent edge), forcing it (or a portion of tile 3) to overlap with the blocker. In order for the tile at location 3 to instead bind to the top left side of the tile at location 2, it would have to overlap with the blocker. This means that p would be blocked and the bit-reader fails, so this case must not be true.

In Figure 4.19b, having the tile at position 2 at the same orientation but an offset above p'_b requires that that tile fill the entire right side of p'_b in order to avoid the yellow tile at p'_α, thus making it collide with the blocker and the bit-reader again fail.

In Figure 4.19c, having the tile at position 2 at the opposite orientation as p'_b but with its southeast corner below the southwest corner of the tile at p'_{end} forces the tiles at positions 2 and 3 to cover all of the left side of p'_b and thus collide with the blocker, once again making the bit-reader fail.

In Figure 4.19d, having the tile at position 2 at the opposite orientation as p'_b with with its southwestern corner above the southwest corner of the tile at p'_{end} again requires that the tiles at positions 2 and 3 to cover the entire left side of p'_b, colliding with the blocker and making the bit-reader fail.

The cases discussed, along with all others which are the same up to rotation, prevent the growth of path p. Therefore, no path from P_1 can grow to a location past the line

$x \leq 0$ without colliding with a previously placed tile or violating the constraints of Definition 4.3.2. (This situation is analogous to a prematurely aborted computation which terminates in the middle of computational step.) This is a contradiction that this is a bit-reader, and thus none must exist.

□ □

The combination of Lemmas 14, 15, 16, and Theorem 6.1 of [13] suffice to prove Theorem 7.

4.8 Bit-reading gadgets

In this section, we give configurations that are then used to construct bit-reading gadgets for 1) single shape systems with regular polygonal tiles with 7 or more sides (See Section 4.8.1.), 2) 2-shaped systems with regular polygonal tiles for pairs of distinct polygons with 3 to 6 sides (See Section 4.8.2.), and 3) single shaped systems with equilateral polygonal tiles with 4, 5, or 6 sides (See Section 4.8.3.). Finally, in Section 4.8.4, we give a bit-reading gadget for a single shaped system with tiles having the shape of an obtuse isosceles triangle. All of the configurations presented here will be used to obtain bit-reading gadgets that read bits from right to left. It should be noted that for all of the polygons considered here, configurations that yield left to right bit-reading gadgets can be obtained by simply reflecting the corresponding right to left configurations.

4.8.1 Single shape systems with regular polygonal tiles

In the following subsections, we give configurations that will be used to construct bit-reading gadgets for single shape systems with regular polygonal tiles with 7 or more sides. While the configurations presented here do not technically fit Definition 4.3.2, we describe how to turn these configurations into bit-reading gadgets that do conform to that definition. In this section, we are concerned with showing how to use the geometry of polygonal tiles to ensure that our bit-reading gadgets properly read and write bits as described in

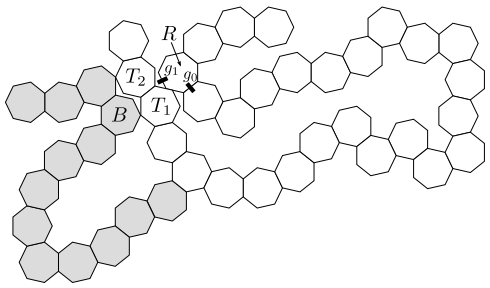
Definition 4.3.2. Therefore, combining the results of this section with Sections 4.3-4.5, we show the following lemma.

Lemma 17. Let P_n be a regular polygon with n sides. Then, for all $n \geq 7$, there exists a single-shaped system $\mathcal{T}_n = (T_n, \sigma_n)$ with shape P_n such that a bit-reading gadget exists for $\mathcal{T}_n$.

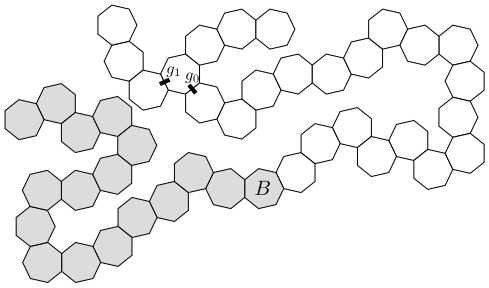
In order to prove Lemma 17, we first consider the cases where n is $7, 8, 9, 13$, or 14, since these cases are handled by giving a specific bit-reading gadget for each case. Second, we give bit-reading gadgets for the cases where n is 10, 11, or 12. These cases are simpler than the former cases and are handled using a more generic approach. Finally, we give the bit-reading gadgets for the cases where $n \geq 15$. These cases are handled by using a single generic scheme for constructing the bit-reading gadgets for each case.

Tiles with $7, 8, 9, 13,$ or 14 sides

In this section we give a description of the bit-reading gadget for heptagonal tiles and give a brief example of a calculation that shows that certain tiles do not overlap. Figure 4.20 gives a depiction of a bit-reader for heptagonal tiles.



(a) A 0 is read, and a 1 cannot be read by mistake since the tile B prevents a heptagonal tile from attaching via the glue labeled g_1.



(b) A 1 is read. This time a 0 cannot be read by mistake since the tile B prevents growth of a path of heptagonal tiles that attach via the glue labeled g_0. Note that some of this path may form, but B prevents the entire path from assembling, and thus prevents a 0 from being read.

Figure 4.20: A connected bit-gadget consisting of heptagonal tiles.

In Figure 4.20, the gray tiles represent the bit writer tiles (representing either 0 or 1), while the white tiles are the bit reader tiles. In our construction, we ensure that we have an assembly sequence such that the gray tiles of a bit-reading gadget bind before any white tiles. Figure 4.20b depicts the case in which a 1 has previously been written and is then read. In this case, we observe that the bit writer tiles prevent the formation of the path of tiles depicted in 4.20a from R to T_1, ensuring that a tile binds to the glue g_1, resulting in a 1 being read. Moreover, since the configuration of Figure 4.20b consists of abutting heptagonal tiles with non-overlapping interiors, we see that with appropriately defined glues, the bit writer configuration and the bit reader configuration are valid assemblies. We can also see that no two tiles of the bit writer configuration and the bit reader configuration have overlapping interiors; this ensures that these two assemblies can be part of the same larger assembly.

Similarly, Figure 4.20a depicts the case in which a 0 has previously been written and is being read. Though much of this configuration consists of abutting heptagonal tiles with non-overlapping interiors, it is not clear that all of the heptagonal tiles have non-overlapping intersection. For example, it is indeed the case that R and T_2 have non-overlapping intersection (It turns out that they do share a portion of an edge.) but it is not clear that the interiors of these tiles do not overlap on some tiny set of points. Moreover, it is not clear that a tile could not attach to the glue g_1. Therefore, we must calculate the distance between these tiles to show that, with appropriately defined glues, the bit reader configuration is a valid assembly, and that no two tiles of the bit writer configuration and the bit reader configuration have overlapping interiors.

Referring to Figure 4.20a, we will first show that the tile labeled R does not prevent the binding of the tile labeled T_1 or the tile labeled T_2. Let c denote the center of the tile R, c_1 denote the center of T_1, and c_2 denote the center of T_2. Then, to calculate c_1 and c_2 relative to c, we assume that R has standard orientation and is centered at the origin. Following the path of tiles from R to T_1 and summing the appropriate roots of unity, we ob-

tain the polynomials $c_1 = \omega^6 - \omega^3 + \omega - \omega^4 + 1 - \omega^4 + \omega - \omega^3 + 1 - \omega^2 + \omega^5 - \omega^2 + \omega^4 - \omega^6 + \omega^4 - \omega^6 + \omega^4 - \omega + \omega^4 - 1 + \omega^3 - \omega^6 + \omega^2$. Note that $c_2 = c_1 - \omega^6$. By simplifying c_1, we get $c_1 = 1 + \omega - \omega^2 - \omega^3 + 2\omega^4 + \omega^5 - 2\omega^6$. Then, as multiplying by ω corresponds to rotating counterclockwise by $2\pi/7$, multiplying by ω^2 rotates Figure 4.20a so that R is still centered at the origin, and the edge of R and the edge of T_1 that appear to overlap in the figure are parallel to the imaginary axis. Hence, to show that R and T_1 do not overlap, it is enough to show that $\mathrm{Re}(\omega^2 c_1) \geq 1$, and to see this, consider the following.

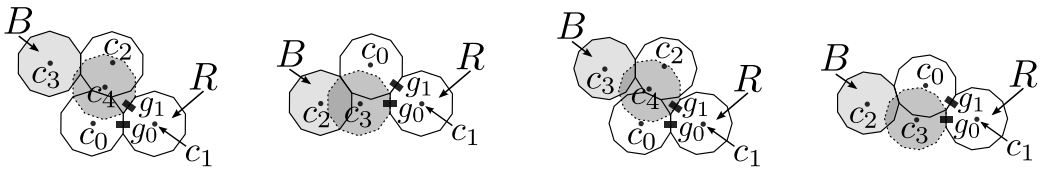
$$\begin{aligned}
\omega^2 c_1 &= \omega^2 + \omega^3 - \omega^4 - \omega^5 + 2\omega^6 + \omega^7 - 2\omega^8 \\
&= \omega^2 + \omega^3 - \omega^4 - \omega^5 + 2\omega^6 + 1 - 2\omega \\
&= \omega^2 + \omega^3 - \omega^{-3} - \omega^{-2} + 2\omega^6 + 1 - 2\omega^{-6} \\
&= 1 + (\omega^2 - \omega^{-2}) + (\omega^3 - \omega^{-3}) + 2(\omega^6 - \omega^{-6})
\end{aligned}$$

Finally, since $(\omega^2 - \omega^{-2})$, $(\omega^3 - \omega^{-3})$, and $2(\omega^6 - \omega^{-6})$ are purely imaginary, we see $\mathrm{Re}(\omega^2 c_1) = 1$. It follows that the intersection of the interiors of R and T_1 is empty. The remainder of the distance calculations are given in the technical appendix. For tiles consisting of regular polygons with $8, 9, 13$, or 14 sides we also give the bit-reading gadgets and calculations in the technical appendix.

Tiles with $10, 11$, or 12 sides

In the cases where tiles consist of regular polygons with $10, 11$, or 12 sides, bit-reading gadgets are relatively simple to construct. Figure 4.21 depicts the configurations that we will use to construct our bit-reading gadgets for each case. Note that since each polygonal tile of these configurations is adjacent to another tile, we need only show that for each configuration depicted in Figure 4.21, of the two exposed glues, g_0 and g_1 of the tile R, a tile can only attach to one of these glues depending on the position of the tile B in the figure. In

other words, for each configuration depicted in Figure 4.21, we show that in the top configuration, B prevents a tile from binding to g_1, and that in the bottom configuration, B prevents a tile from binding to g_0.



(a) Bit-reading gadget configuration for decagonal tiles.

(b) Bit-reading gadget configuration for hendecagonal tiles.

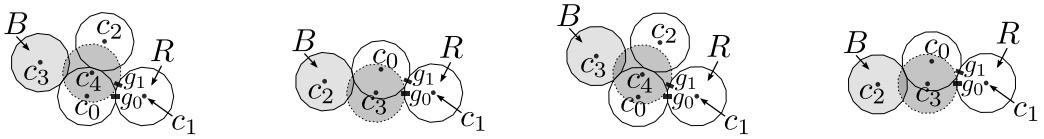
(c) Bit-reading gadget configuration for dodecagonal tiles.

Figure 4.21: (a), (b) and (c) each depict two configurations of polygonal tiles which represents either a 0 (bottom) or a 1 (top).

Like the bit-reading gadgets themselves, the calculations used to show the correctness of these bit-reading gadgets are relatively simple when compared to the previous cases. For example, for decagonal tiles, in top configuration depicted in Figure 4.21a, to show that B prevents a tile from binding to g_0, note that a polygon centered at c_2 and a polygon centered at c_3 overlap. Let ω be the 10^{th} root of unity $e^{\frac{2\pi i}{10}}$. Note that relative to c_2, $c_3 = \omega + \omega^9 - 1$. Hence, $c_3 = \omega + \omega^9 - 1 = 2\,\mathrm{Re}(\omega) - 1 = 2\cos\left(\frac{2\pi}{10}\right) - 1$. Then the distance d from c_3 to c_2 satisfies $d = |2\cos\left(\frac{2\pi}{10}\right) - 1| < .62$, and therefore the intersection of the interiors of a decagon centered at c_3 and a decagon centered at c_2 is nonempty. Hence, a decagonal tile cannot bind to the glue g_0. The remaining calculation for the decagonal tiles case as well as the calculations for the hendecagonal and dodecagonal cases are given in Section 4.10.1.

Tiles with 15 or more sides

In the cases where tiles consist of regular polygons with 15 or more sides, we give a general scheme for obtaining bit-reading gadgets for each case. Figure 4.22 depicts the bit-reading gadgets for each case. For the top configurations of Figure 4.22, note that since each polygonal tile of these bit-reading gadgets is adjacent to another tile, we need only show that for each top configuration depicted in Figure 4.22, of the two exposed glues, g_0 and g_1 of the tile R, B prevents a tile from binding to g_0. In the bottom configurations of Figure 4.22, we not only need to show that B prevents a tile from binding to g_1, but we must also show that B does not prevent a tile (the tile centered at c_2 in the bottom configurations for Figure 4.22) from binding to the tile that binds to g_0. The latter statement ensures that when we use the bit-reading gadgets obtained from these configurations to simulate a Turing machine, in the case that a 0 is read by attaching a tile to g_0, B does not prevent further growth of an assembly.



(a) Bit-reading gadget configuration for pentadecagonal tiles.

(b) Bit-reading gadget configuration for hexadecagonal tiles.

(c) Bit-reading gadget configuration for heptadecagonal tiles.

Figure 4.22: (a), (b) and (c) each depict two configurations of polygonal tiles which represents either a 0 (bottom) or a 1 (top).

Now, consider a polygon P_n with $n \geq 15$ sides and let ω be the n^{th} root of unity $e^{\frac{2\pi i}{n}}$. Then, the general scheme for constructing a bit-reading gadget falls into two cases. First, if n is odd (the cases where n is even are similar), relative to a tile with negated orientation (the polygon labeled R in the configurations in Figure 4.22), the two configurations that

give rise to the bit-reading gadget are as follows. Let k be such that $n = 2k + 1$ ($n = 2k$ if n is even). Referring to the top configurations of Figure 4.22. To "write" a 1, the configuration is obtained by centering a blocker tile with negated orientation, labeled B in the top configurations of Figure 4.22, at $-\omega^{n-1} + \omega^{k+1}$ (whether n is even or odd) relative to R. Then to "read" a 1, R exposes two glues g_1 and g_0 such that if a tile binds to g_1, it will have standard orientation and be centered at $-\omega^{n-1}$ (whether n is even or odd) and if a tile that binds to g_0, it will have standard orientation and be centered at -1. We will show that B will prevent this tile from binding. This gives the configuration depicted in the top figures of Figure 4.22. Now, referring to the bottom configurations of Figure 4.22, to "write" a 0, the configuration is obtained by centering a blocker tile with negated orientation, labeled B in the bottom configuration of Figure 4.22a, at $-1 + \omega^{k-1}$ ($-1 + \omega^{k-2}$ if n is even) relative to R. In this case, we will show that B prevents a tile from binding to g_1. In addition, we place a glue on the tile that binds to g_0 that allows for another tile to bind to it so that its center is at $c_2 = -1 + \omega^{\lfloor \frac{k-1}{2} \rfloor}$ ($c_2 = -1 + \omega^{\frac{k-2}{2}}$ if n is even) relative to R. This gives the configuration depicted in the bottom figures of Figure 4.22a and Figure 4.22c. Moreover, we show that neither R nor B prevent the binding of this tile.

In order to perform the calculations used to show the correctness of these bit-reading gadgets, we consider the cases where n is even and where n is odd. Here we give brief versions of the calculations when n is odd. For more detail and calculations for the case where n is even, see the technical appendix.

Suppose that $n = 2k+1$. We now refer to the bottom configurations of Figure 4.22a. To show that a polygon centered at c_1 and a polygon centered at c_2 do not overlap, consider the case where k is odd (the case where k is even is similar). Note that relative to c_0, $c_1 = 1$ and $c_2 = \omega^{\frac{k-1}{2}}$. Then the distance d_n from c_1 to c_2 satisfies the following equation.

$$d_n^2 = \left(1 - \cos\left(\frac{(k-1)\,\pi}{n}\right)\right)^2 + \sin^2\left(\frac{(k-1)\,\pi}{n}\right)$$

Substituting $k = \frac{n-1}{2}$ for k and simplifying, we obtain $d_n^2 = 2 + 2\sin\left(\frac{3\pi}{2n}\right)$. It is well

known that for regular polygons with n sides and apothem $\frac{1}{2}$, the circumradius is given by $\frac{1}{2\cos\left(\frac{\pi}{n}\right)}$. Hence, to show that a polygon centered at c_1 and a polygon centered at c_2 do not overlap, we show that $d_n^2 > \frac{1}{\cos^2\left(\frac{\pi}{n}\right)}$ for $n \geq 15$. To see this, note that $\cos^2\left(\frac{\pi}{n}\right) d_n^2 = 2\cos^2\left(\frac{\pi}{n}\right)\left(1+\sin\left(\frac{3\pi}{2n}\right)\right)$. Then for $n \geq 15$, $2\cos^2\left(\frac{\pi}{n}\right)\left(1+\sin\left(\frac{3\pi}{2n}\right)\right) > 2\cos^2\left(\frac{\pi}{4}\right) = 1$. It then follows that $d_n > \frac{1}{\cos\left(\frac{\pi}{n}\right)}$. Therefore, d_n is greater than twice the circumradius of our polygons. Hence, a polygon centered at c_1 and a polygon centered at c_2 do not overlap.

To show that a polygon centered at c_3 and a polygon centered at c_4 overlap, note that relative to c_1, $c_3 = -1 + \omega^{k-1}$ and $c_4 = -\omega^{n-1}$. Therefore, the distance d_n from c_3 to c_4 is satisfies the equation

$$\begin{aligned} d_n^2 &= \left(-1+\cos\left(\frac{2(k-1)\pi}{n}\right)+\cos\left(\frac{2(n-1)\pi}{n}\right)\right)^2 \\ &= \quad +\left(\sin\left(\frac{2(k-1)\pi}{n}\right)+\sin\left(\frac{2(n-1)\pi}{n}\right)\right)^2 \end{aligned}$$

Substituting $k = \frac{n-1}{2}$ for k and simplifying, we obtain,

$$d_n^2 = 1 + 2\left(2\sin^2\left(\frac{\pi}{n}\right)\left(1-2\cos\left(\frac{\pi}{n}\right)\right)\right).$$

Note that for each $n > 2$, $d_n^2 < 1$. To see this, it suffices to show that

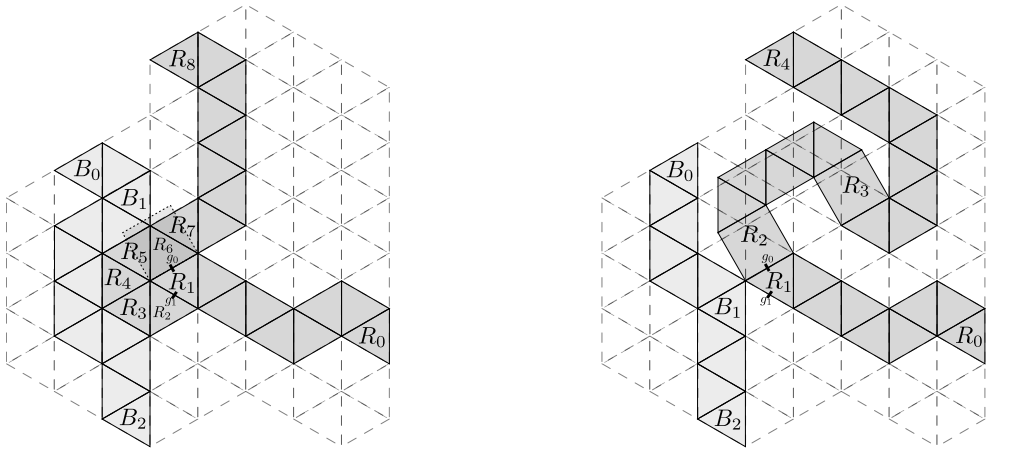
$$2\sin^2\left(\frac{\pi}{n}\right)\left(1-2\cos\left(\frac{\pi}{n}\right)\right) < 0.$$

This follows from the fact that $2\sin^2\left(\frac{\pi}{n}\right) > 0$ and $1 - 2\cos\left(\frac{\pi}{n}\right) < 0$ for $n > 2$. Now, since for each $n > 2$, $d_n^2 < 1$, we see that $d_n < 1$. Since the length of the apothem for each tile is assumed to be $\frac{1}{2}$, we can conclude that a polygon centered at c_3 and a polygon centered at c_4 must overlap.

4.8.2 2-shaped systems with regular polygonal tiles

In this section we describe bit-reading gadgets for 2-shaped systems whose tileset consists of two distinct regular polygons. We assume that the edges of all polygonal tiles have the same length. The bit-reading gadgets that we give here are normalized on-grid bit-readers.

Lemma 18. Let P_n and Q_m be a regular polygons with n and m sides of equal length. Then, for all $n \geq 3$ and $m \geq 3$ such that $n \neq m$, there exists a 2-shaped system $\mathcal{T}_{n,m} = (T_{n,m}, \sigma_{n,m})$ with shapes P_n and Q_m such there a bit-reading gadget exists for $\mathcal{T}_{n,m}$.



(a) A 1 is read. This time a 0 cannot be read by mistake since the tile B prevents growth of a path of tiles that attach via the glue labeled g_0.

(b) A 0 is read, and a 1 cannot be read by mistake since the tile B prevents a tile from attaching via the glue labeled g_1.

Figure 4.23: A connected bit-gadget consisting of tiles shaped like a regular triangle or square.

(a) and (b) of Figure 4.23 depict bit-reading gadgets which give a scheme for "writing a bit" as growth proceeds from left to right, and "reading a bit" as growth proceeds from right to left. To write a bit, we can define unique glues that enforce the assembly of the path of tiles (light gray tiles in (a) and (b) of Figure 4.23) starting from the tile labeled B_0 and ending at a tile labeled B_2 in (a) and (b). Assuming that the light gray tiles are part of an existing assembly, to read a bit, we define unique glues that enforce the (dark gray tiles in (a) and (b)) starting from the tile labeled R_0 and ending with the tile labeled R_1. Then, R_1 exposes two glues labeled g_0 and g_1 in Figure 4.23. Now, depending on whether

an assembly which represents a 0 is present or an assembly which represents a 1 is present, either a triangular tile with a glue labeled g_1 binds to R_1 via the glue g_1 exposed by R_1 (depicted in Figure 4.23a) or a square shaped tile with a glue labeled g_0 binds to R_1 via the glue g_0 exposed by R_1 (depicted in Figure 4.23b).

In the former case, denote the triangular tile which binds to R_1 by R_2; this tile is labeled R_2 in Figure 4.23a. Then, we can define glues that allow the tiles R_i for $i = 3, 4, 5, 6$ or 7 to bind in that order as depicted in Figure 4.23a. Finally, we define a set of tiles that form the path of tiles from R_7 to R_8. The latter case, depicted in Figure 4.23b, is similar. In this case, a square tile (labeled R_2) binds to g_0. We define this tile such that the path of tiles from R_2 to R_4 assembles. Note that the square tile labeled R_3 ensures that the triangular tiles along this path of tiles from R_3 to R_4 are on-grid. In particular, R_4 is on-grid. Lastly, we refer to each configuration in Figure 4.23 and note that relative to the underlying grid (shown as dashed lines), B_0, B_2, R_0 and R_8 in (a) and respectively B_0, B_2, R_0 and R_4 in (b) are on-grid and in the same location. It is straightforward to see that such configurations can be extended to give a normalized on-grid bit-reading gadget that conforms to Definition 4.3.2.

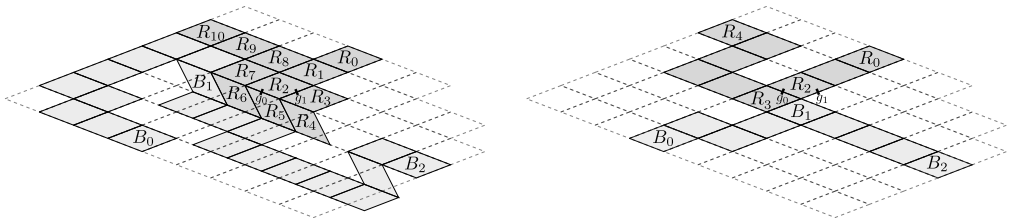
Constructions for normalized on-grid bit-reading gadgets for pairs of regular polygons with sides m and n where $3 \leq m \leq 6$, $5 \leq n \leq 6$ and $m \neq n$ are similar and are given in Section 4.10.2.

4.8.3 Single shaped systems with equilateral polygonal tiles

In this section we describe bit-reading gadgets for single-shaped systems whose tile set consists of an equilateral polygon. The bit-reading gadgets that we give here are normalized on-grid bit-readers. Note that for polygons with 7 or more sides, Lemma 17 implies the following lemma. Hence, we need only show Lemma 19 for equilateral polygons with 4, 5, or 6 sides. Therefore, we give normalized on-grid bit-reading gadgets for all three cases showing the following lemma. It should be noted that the general grid construction given in Section 4.5 pertain to regular polygons. Similar techniques can be used to obtain grids

for the equilateral polygonal tiles in this section. The grids themselves are depicted using dashed lines in the figures of this section.

Lemma 19. For all $n \geq 4$, there exists an equilateral polygon P_n with n sides and a single-shaped system $\mathcal{T}_n = (T_n, \sigma_n)$ with shape P_n such a bit-reading gadget exists for $\mathcal{T}_n$.



(a) A 0 is read, and a 1 cannot be read by mistake since the tile B prevents a quadrilateral tile from attaching via the glue labeled g_1.

(b) A 1 is read. This time a 0 cannot be read by mistake since the tile B prevents growth of a path of quadrilateral tiles that attach via the glue labeled g_0.

Figure 4.24: A connected bit-gadget consisting of quadrilateral tiles. This figure also depicts the tile shape.

(a) and (b) of Figure 4.24 depict bit-reading gadgets which give a scheme for "writing a bit" as growth proceeds from left to right, and "reading a bit" as growth proceeds from right to left. To write a bit, we can define unique glues that enforce the assembly of the path of tiles (light gray tiles in (a) and (b) of Figure 4.24) starting from the tile labeled B_0 and ending at a tile labeled B_2 in (a) and (b). Assuming that the light gray tiles are part of an existing assembly, to read a bit, we define unique glues that allow R_0, R_1 and R_2 to bind in that order. Then, R_2 exposes two glues labeled g_0 and g_1 in Figure 4.24. Now, depending on whether an assembly which represents a 0 is present or an assembly which represents a 1 is present, either a quadrilateral tile with a glue labeled g_1 binds to R_2 via the glue g_1 exposed by R_2 (depicted in Figure 4.24a) or a quadrilateral tile with a glue labeled g_0 binds to R_2 via the glue g_0 exposed by R_2 (depicted in Figure 4.24b). In the former case, denote the quadrilateral tile which binds to R_2 by R_3; this tile is labeled R_3 in Figure 4.24a. Then, we can define glues that allow the tiles R_i for $3 \leq i \leq 10$ to bind in that order as depicted in Figure 4.24a. Moreover, we note that B_1 prevents a tile from binding to g_0. The latter case, depicted in Figure 4.24b, is similar. In this case, a quadrilateral tile

(labeled R_2) binds to g_0. We define this tile such that the path of tiles from R_2 to R_4 assembles. In the case of Figure 4.24b, we also note that B_1 prevents a tile from binding to g_1. Finally, we note that relative to the underlying grid (shown as dashed lines in (a) and (b) of Figure 4.24) this configuration can then be used to obtain a normalized on-grid bit-reading gadget.

Constructions for normalized on-grid bit-reading gadgets for equilateral polygons with sides 5 or 6 sides are similar and are given in Section 4.10.3.

4.8.4 A single shaped system with triangular tiles

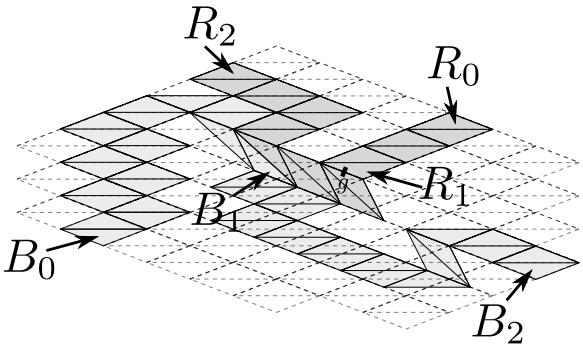
In this section we describe bit-reading gadgets for single-shaped systems whose tile set consists of a particular obtuse isosceles triangle. We assume that the edges of all triangular tiles have the same length. The bit-reading gadgets that we give here are normalized on-grid bit-readers. Again, it should be noted that while the general grid construction given in Section 4.5 pertain to regular polygons. Similar techniques can be used to obtain grids for the triangular tiles in this section. The grids themselves are depicted using dashed lines in the figures of this section.

Lemma 20. There exists an obtuse isosceles triangle P and a single-shaped system $\mathcal{T} = (T, \sigma)$ with shape P such a bit-reading gadget exists for $\mathcal{T}$.

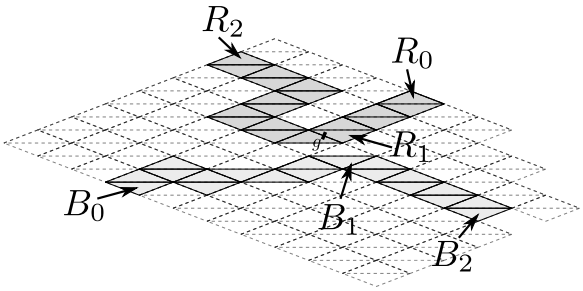
Figure 4.25 depicts the configurations that give rise to bit-reading gadgets for single-shaped systems whose tiles have the shape of an obtuse isosceles triangle. As in Section 4.8.3, one can see that these configurations can be used to obtain a normalized on-grid bit-reading gadget.

4.9 Building *normalized* bit-reading gadgets

Let P be a regular polygon with 7 or more sides, and let g_α denote the terminal assembly of the tile system given in Lemma 13. We now show that given a bit-reading gadget from



(a) A configuration of polygonal tiles which represents a 0



(b) A configuration of polygonal tiles which represents a 1.

Figure 4.25: Bit-reading gadget configuration for tiles with the shape of an irregular triangle.

proceeding section corresponding to the regular polygon P, we can form an on grid bit-reading gadget (with respect to g_α). In order to show this, we first show how the individual bit-writers can be grown in an on grid manner (with the bit reader that reads these writers also on grid), and then we show how to find positions common to these bit-writers so that up to translation, the bit writers start and end in the same place. Before we begin our construction, we introduce a couple of definitions. We denote the location of the center of a tile P in the complex plane by $c(P)$. We say that a tile P is *x-centered on grid* g_α provided that $c(P) = c(P')$ and P and P' have the same orientation for some tile $P' \in g_\alpha$.

At a high-level, we construct a normalized bit-reading gadget from one of the gadgets presented in Section 4.8 in the following way. Consider Figures 4.26 and 4.27 where normalized bit-reading gadgets are given in the case of heptagonal tiles. In those figures a bit is written from west to east and read from east to west. When writing a bit and starting from the southwest-most black tile, assembly proceeds via attachment of a single tile at a time on some fixed grid g_α (shown in the background in the figures as white heptagons). Then, the blue tiles "shift" off this grid and onto another grid, g'_α say. This shifting ensures that the tile labeled R in those figures is on the grid g_α. Then, the portion of a bit-reading gadget which encodes a 0 (Figure 4.26) or 1 (Figure 4.27) is assembled. The tiles which make up this portion are purple in the figures. Call the set of these tiles S. At this point, we are possibly on a grid g''_α which may or may not be distinct from g_α or g'_α. Finally, we "shift" back onto the grid g_α by assembling the remaining portion of a bit-reading gadget (those tiles of the bit-reading gadget that are not in S). The tiles which produce this final shift are green in Figures 4.26 and 4.27. At this point, a path of tiles (each of which is on g_α) assemble until the southeast-most black tile in the figures attaches. This bit is read using the orange, red and yellow tiles. The tile R is on grid g_α. The red tiles are the path of unblocked tiles whose assembly indicates that the appropriate bit is read. The final red tile that is placed may not be on grid g_α. Therefore, the yellow tiles (whose assembly sequence is essentially that of the red tiles in reverse order) "shift" back onto grid g_α. Note that the

black tiles and the end tiles of the reading path of tiles (the orange, yellow, and red tiles) in Figure 4.26 have locations that match the locations of the respective tiles in Figure 4.27. This ensures that we can "plug" these gadgets into a zig-zag growth pattern to simulate a Turing machine.

4.9.1 Constructing on grid bit-writer configurations

The α_0 on grid bit writer will consist of three parts which we call: 1) a blocker subconfiguration, 2) an east shifting subconfiguration and 3) a west shifting subconfiguration. The three subconfigurations are all formed by modifying a "base" configuration which we describe now. The base configuration is formed by modifying the assembly obtained when the bit-reading gadget described in Section 4.8 "reads a 0".

If the bit-reading gadget for P is simple (e.g. those shown in Figures 4.21 and 4.8.1), we first extend the bit writing portion of the gadget in the following way. To begin, observe that the bit writer portion of the bit reading gadget will consist of a tile with negated orientation which we will call B. Note that by the construction of these simple bit reading gadgets, we can always place a tile which has standard orientation at a position of $\omega^{\lceil \frac{3k}{4} \rceil}$ relative to B. After placing this tile, we continue placing tiles so that we form a path of tiles from B such that the last tile placed in this path is the northernmost tile in the bit reading gadget configuration and has standard orientation (see Figure 4.28b). Next we grow a path of tiles from B that extends south so that the last tile placed in this path is the southernmost tile in the bit reading configuration as shown in Figure 4.28b.

We say that the first tile to be placed in the bit-writer subconfiguration of the bit-reading gadget is the northernmost "end tile" in the path. The other "end tile" in the path we refer to as the last tile to be placed in the bit-writer subconfiguration. Also, recall that the tile R in the bit reading gadget is the tile from which the bit reader grows. To construct the base configuration, we simply remove the tiles in the configuration which do not lie on either the path from the first tile in the bit-writer to the tile R or the path from the last tile in the bit-writer to the tile R. Furthermore, we extend a path from the first tile

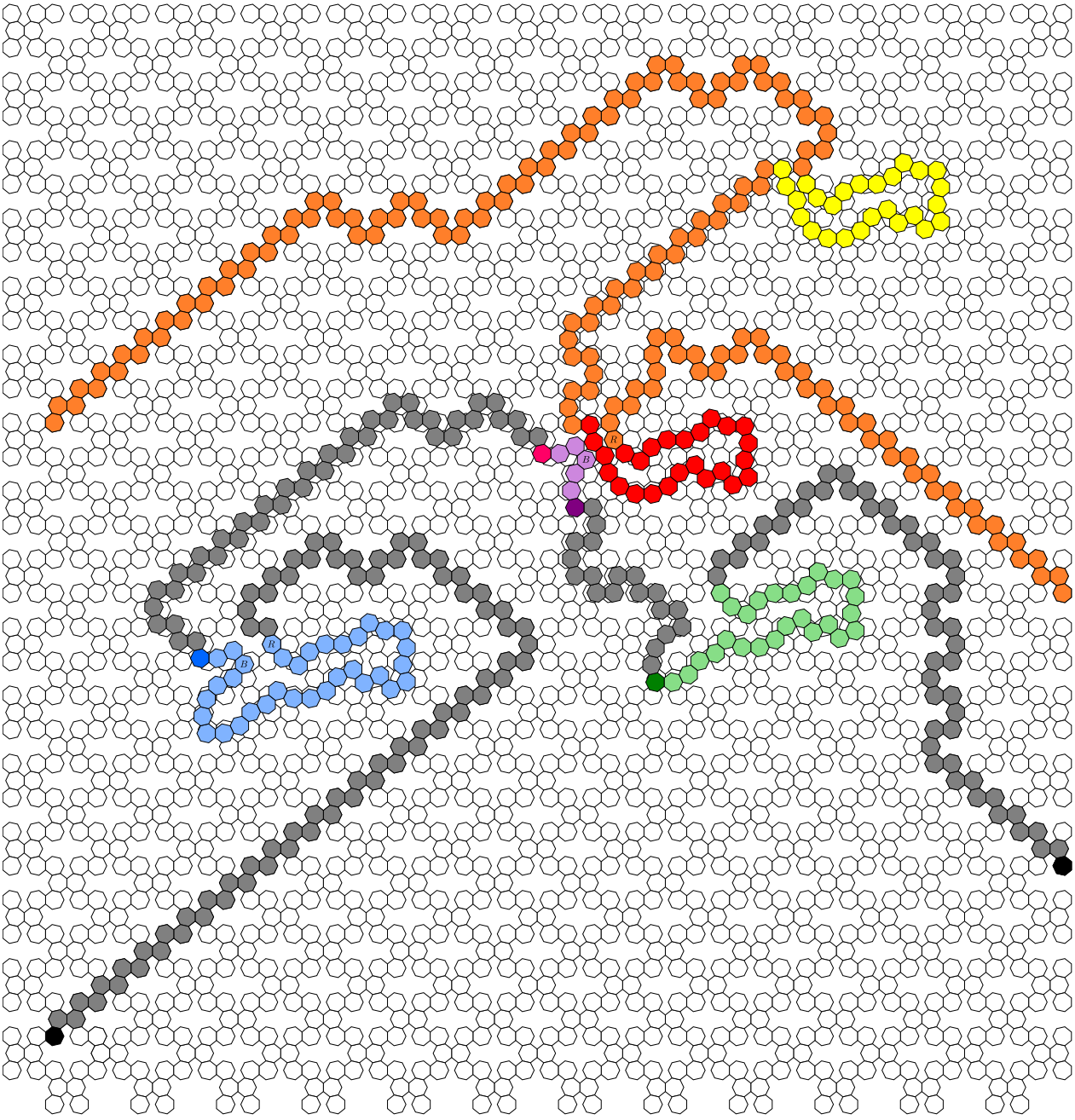


Figure 4.26: The completed bit-reading gadget for heptagons when a 0 is "read". The grey tiles represent paths which connect the subconfigurations in the bit writer. The blue tiles represent C_{α_w}, and the dark blue tile represents t_{ww}. The purple tiles represent C_α, and the dark purple tile represents t_s. The pink tile represents t_w. The green tiles represent C_{α_e}, and the dark green tile represents t_{se}. All other color of tiles represent tiles composing the bit reader. In this figure the bit is written from west to east and read from east to west.

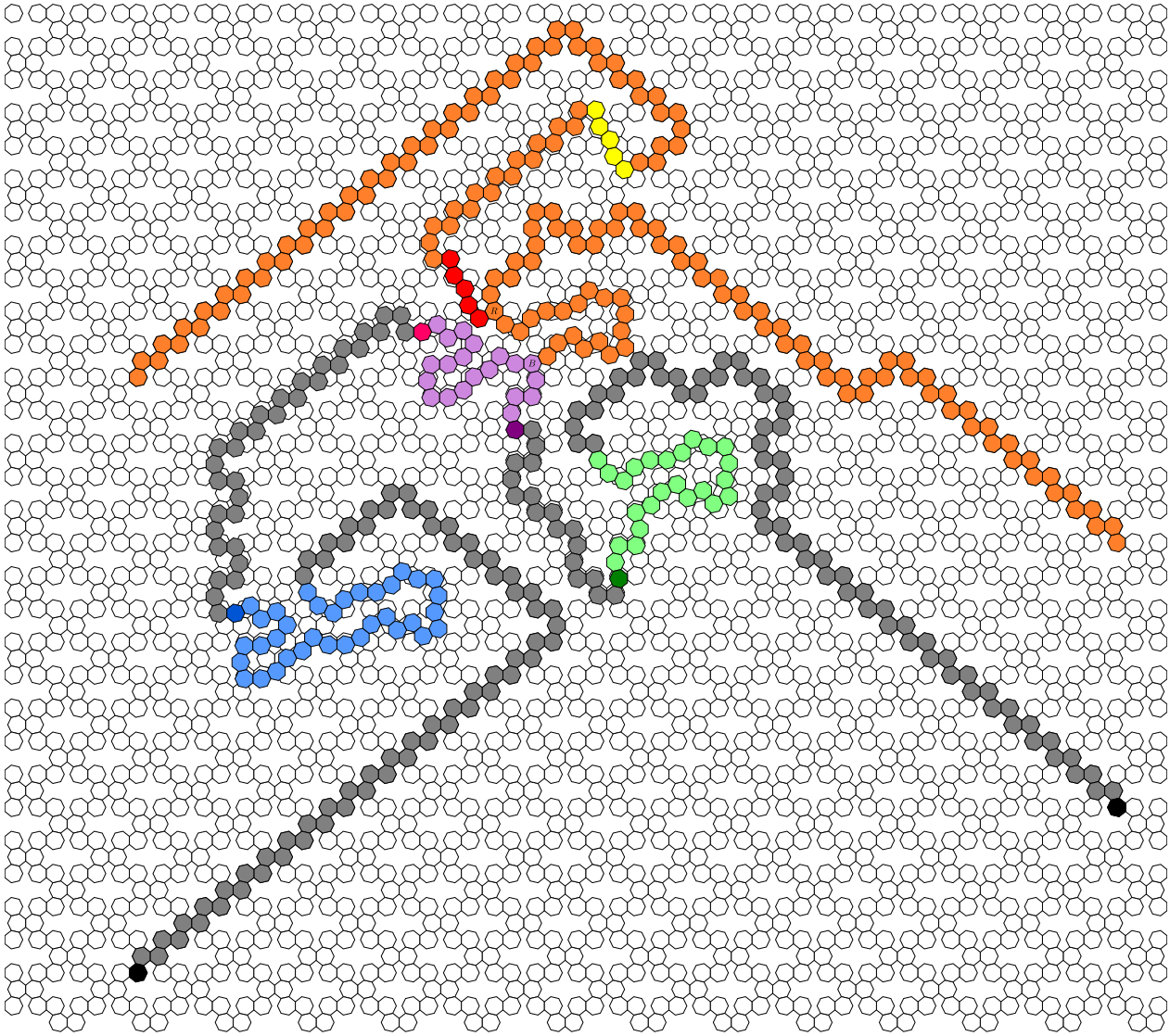
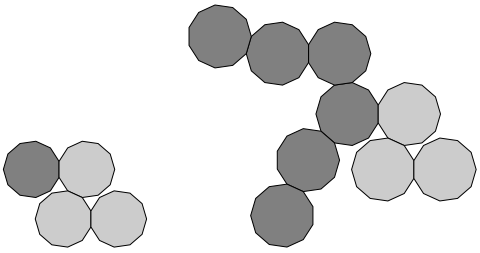


Figure 4.27: The completed bit-reading gadget gadget for heptagons when a 1 is "read". The grey tiles represent paths which connect the subconfigurations in the bit writer. The blue tiles represent C_{α_w}, and the dark blue tile represents t_{ww}. The purple tiles represent C_α, and the dark purple tile represents t_s. The pink tile represents t_w. The green tiles represent C_{α_e}, and the dark green tile represents t_{se}. All other color of tiles represent tiles composing the bit reader. In this figure the bit is written from west to east and read from east to west.



(a) A simple bit reading gadget formed from a regular polygon with 10 sides which has read a 0 (also shown in Figure 4.21.

(b) Extending the bit reading gadget to form our base configuration.

Figure 4.28: A simple bit reading gadget (which "read a 0") and its extension (which will form our base configuration). The darkly shaded tiles are the bit writer portion of the bit reading gadget.

to be placed in the bit-writer portion of the bit-reading gadget so that the last tile placed in this path has negated orientation and is the westernmost tile in the bit reading gadget configuration. Call this configuration C_α.

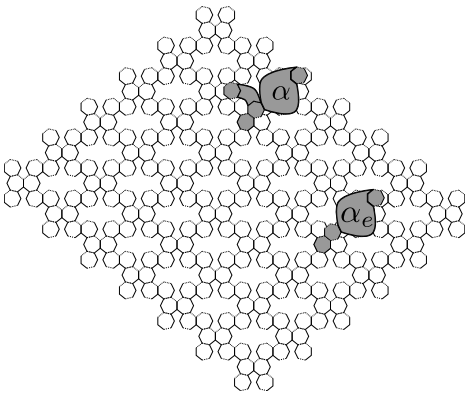
Let the tile t_s represent the westernmost tile of the set of southernmost tiles in the bit-writer portion of the configuration C_α. We consider two cases: 1) the tile t_s has negated orientation and 2) the tile t_s has standard orientation. In case 1, we add a tile in standard orientation to the configuration at location $-\omega^{\lfloor \frac{k}{4} \rfloor}$ relative to the tile t_s. We know that this is still a valid configuration by the construction of the bit reading gadgets in the previous section and the assumption that t_s is the westernmost tile of the set of southernmost tiles. Note that after this modification we are now in case 2. In the case that t_s has standard orientation, we translate C_α so that the tile t_s is 1-centered on the grid g_α. We denote the bounding box of C_α by B_α and the dimensions of B_α by $m_B \times n_B$.

Now, let C_{α_e} be the configuration obtained by taking a copy of C_α and removing all tiles which do not lie on the shortest path from t_s to R. For clarity we denote the tile t_s in

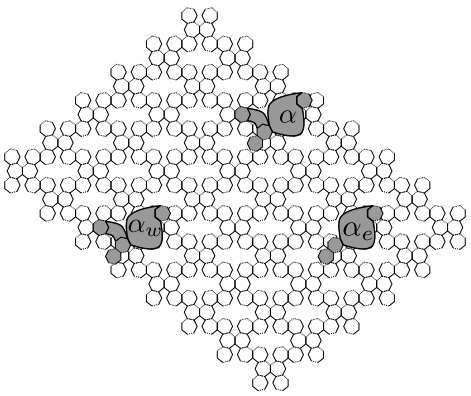
C_{α_e} by t_{se}. Translate this configuration so that it has the following properties: 1) the tile t_{se} is 1-centered on the gird g_α, 2) $\mathrm{Re}(c(t_{se})) - \mathrm{Re}(c(t_s)) \geq 5$, and 3) $\mathrm{Im}(c(t_s)) - \mathrm{Im}(c(t_{se})) \geq n_B$.

Define the tile t_w to be the westernmost tile of C_α. Translate the configurations C_α and C_{α_e} so that they remain in the same positions relative to each other and the tile t_w is 4-centered on the grid g_α. We now make a copy of configuration C_α, which we call C_{α_w}, and denote the tile t_w in C_{α_w} by t_{ww}. We translate the configuration C_{α_w} so that its location meets the following requirements: 1) the tile t_{ww} is 4-centered on the grid g_α, 2) $\mathrm{Re}(c(t_w)) - \mathrm{Re}(c(t_{ww})) \geq n_{B'} + 5$, and 3) $\mathrm{Im}(c(t_w)) - \mathrm{Im}(c(t_{ww})) \geq m_{B'} + 5$. Call this configuration C''.

The blocker subconfiguration consists of a modified version of the configuration C_α. Namely, it consists of the configuration C_α with all the tiles which do not lie on the minimal path from t_w to t_s removed. We leave these extra tiles in the figures in the hopes that it will make the proof of correctness clearer. The east shifting subconfiguration is given by C_{α_e} and the west shifting subconfiguration is given by C_{α_w}.



(a) Positioning the configuration α_e relative to the configuration α.



(b) Positioning the configuration α_v relative to the configuration α.

Figure 4.29: The three configurations and their positions relative to each other.

4.9.2 Connecting the bit-writer subconfigurations

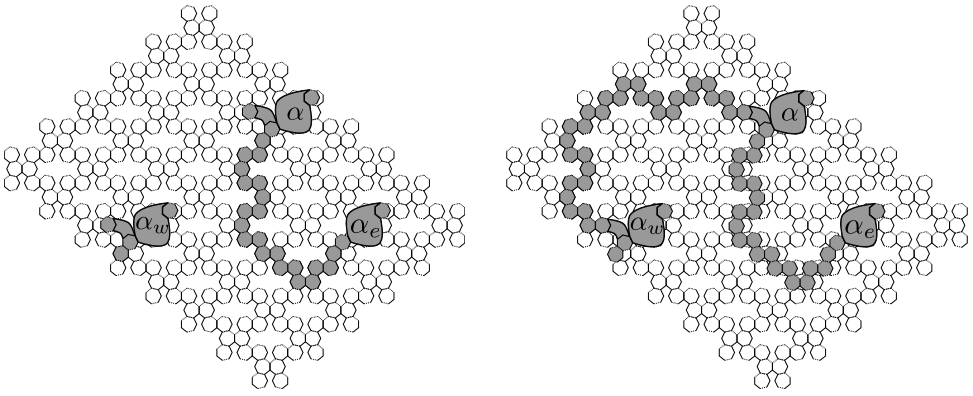
Intuitively, we connect the blocker configuration to the east shifting configuration in the following way. We shift the three configurations so that they remain in the same positions relative to each other, and the tile t_s is 1-centered. Note that by construction, the tile t_{se} lies at least 5 tiles to the right of t_s. Thus, we can grow an almost straight line of tiles, which all lie on grid, until there is a tile which lies south of t_s in the path. Call this path p_{se}. We then grow a path on grid from the last tile placed in p_{se} that attaches to the southernmost side of the tile t_s. An example of this can be found in Figure 4.30a.

Similarly, to attach the blocker configuration to the west shifting configuration, we first shift the two unconnected configurations (note there are now only two unconnected configurations now since the blocker configuration and the east shifting configuration are now attached) so that they remain in the same positions relative to each other, and the tile t_w is 4-centered on the grid. Then we grow an on grid path of tiles from t_n to the west (while keeping the path as straight as possible) until the path has tiles which lie to the west of $t_n w$ at which time the path turns (while still on grid) and grows south until it attaches to $t_n w$. An example of this can be found in Figure 4.30b.

More formally, to connect the configurations C_α to C_{α_e} in the configuration C'', we grow a path in the following manner. The first tile is placed with negated orientation and 3-centered so that it completely shares a common edge with t_s. We then grow a periodic path of tiles to the south with the tiles in the same positions and grid locations as the path of tiles in Figure 4.30a. This repeats until a 1-centered tile is placed so that it has the same imaginary part as tile t_{se}. Once this occurs, we grow a periodic path of appropriately positioned tiles to the east in the same positions and grid locations as the path of east growing tiles in Figure 4.30a. We do this until the 2-centered tile completely shares an edge with the tile t_{se} as shown in Figure 4.30a. We call this configuration C_e.

To connect the configurations C_α to C_{α_w} in C_e, a path is grown from C_α to C_{α_w} as follows. First, we shift the configuration C_e so that the tile t_w is 4-centered. Note that this

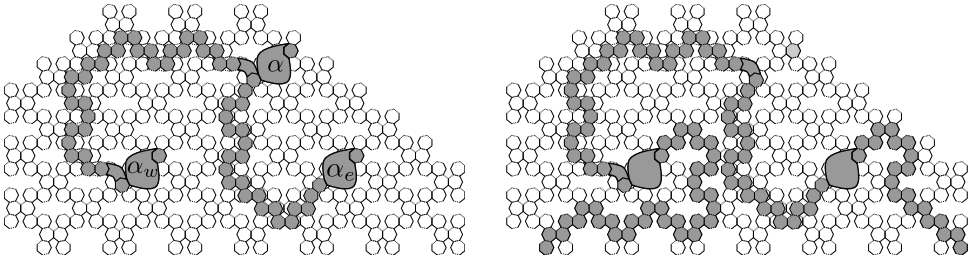
also means the tile $t_w w$ is also 4-centered. We then grow a periodic path of 1-centered, 2-centered, 6-centered, 4-centered, 1-centered, 3-centered, 5-centered, and 4-centered tiles to the west (as shown in Figure 4.30b) until a 4-centered tile is placed so that it has the same real part as the tile t_{ww}. Once this occurs, we grow the periodic pattern south shown in Figure 4.30b until the 1-centered tile in our path completely shares a common edge with t_{ww}. Call this configuration C'.



(a) Connecting C_{α_e} to C_α.

(b) Connecting C_{α_w} to C_α.

Figure 4.30: Connecting the configurations.



(a) Positioning the configurations so that the R tile is on grid.

(b) Growing the final paths of the bit writer configuration and removing the non-bit-writer portion of the configuration α. We refer to this configuration as the α_0 bit-writer.

Figure 4.31: Completing the bit writer.

Now we describe how to grow out "arms" from the bit writer that are on grid which will allow the bit writers to connect to each other. First, we translate the configuration C' so that the tile R in C_α is 4 centered. Note that this will imply the R tiles in the configurations C_{α_e} and C_{α_w} are also 4-centered as shown in Figure 4.31a. We then place a tile

such that it has negated orientation, 3-centered, and both the southernmost tile and east most tile in the configuration C' (call this tile t_{wb}). Likewise, we also place a tile such that it has standard orientation, it is 6-centered, and both the southernmost tile and easternmost tile in the configuration C' (call this tile t_{eb}). Next, we place tiles on grid so that a path of tiles is formed from the R tile in C_{α_w} to the 3-centered tile placed above as shown in Figure 4.31b. Similarly, we place tiles on grid so that a path of tiles is formed from the R tile in C_{α_e} to the 6-centered tile placed above. Call this configuration C.

We construct α_1 in a manner similar to our construction of α_0. The only difference in our construction of α_1 will be that the configuration obtained from the bit reading gadget "reading a 0" will be used as our base configuration.

4.9.3 Normalizing bit-writers

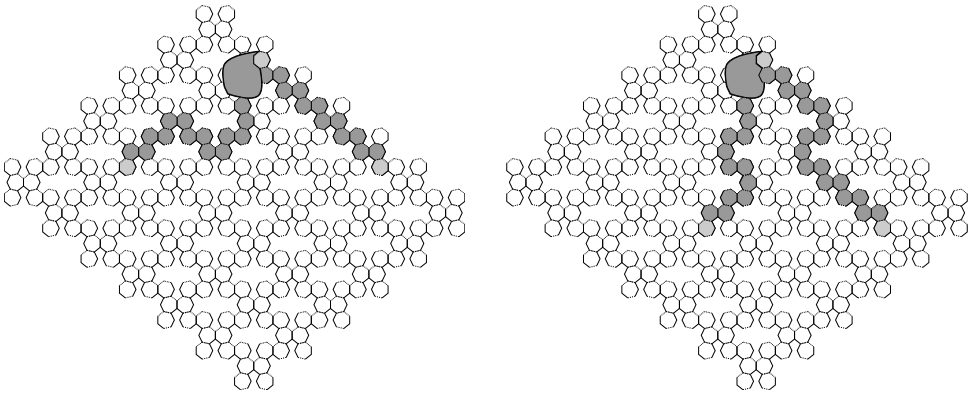
Now that we have constructed on grid bit reading gadgets, we can describe the construction of normalized bit-writers.

Construction of the normalized bit-writers begins by laying down the configurations C_{α_0} and C_{α_1} in the plane so that the tile labeled R in each configuration (see above for the description of R) lies centered at the same point. Next, we remove all tiles in the two configurations except for the tiles t_{wb} and t_{eb} in each bit writer. We now place two new extremal tiles. The first tile we place should be both the southernmost and westernmost tile in the configuration as well as 3-centered. Denote this tile $t_{\max w}$. The location of the second tile's center should have the same imaginary part as the location of the center of the tile $t_{\max w}$. In addition, this tile, which we denote $t_{\max e}$ should be the easternmost tile in the configuration. See Figure 4.33 for an example.

Now, we consider the configuration obtained above with all tiles contributed by C_{α_1} removed. We place a connected path of tiles from the tile t_{wb} to the tile $t_{\max w}$ as shown in Figure 4.34a. Note that this path of tiles is such that none of the interior points of tiles overlap and every tile is connected to some other tile in the path by a completely shared edge. Similarly, we place tiles so as to form a path from the tile t_{eb} to the tile $t_{\max w}$ which

is also shown in Figure 4.34a. These paths of tiles are then attached to configuration C_{α_0} in the same manner they are attached to the extremal tiles in the current configuration to form the configuration for the normalized α_0 bit writer (shown in Figure 4.35a.

We also repeat this same process for the configuration obtained by considering the configuration in Figure 4.33 with all tiles contributed by C_{α_0} removed which yields Figure 4.34b. After "copying and pasting" these paths, we obtain the configuration for the normalized α_1 bit writer which is shown in Figure 4.35b.



(a) An example schematic of an α_0 bit-writer.

(b) An example schematic of an α_1 bit-writer.

Figure 4.32: Completing the bit writer.

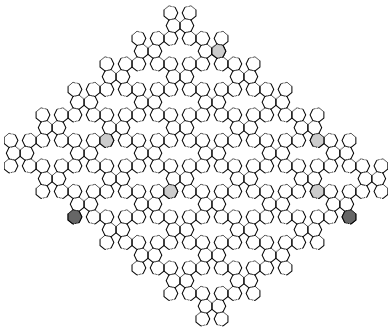
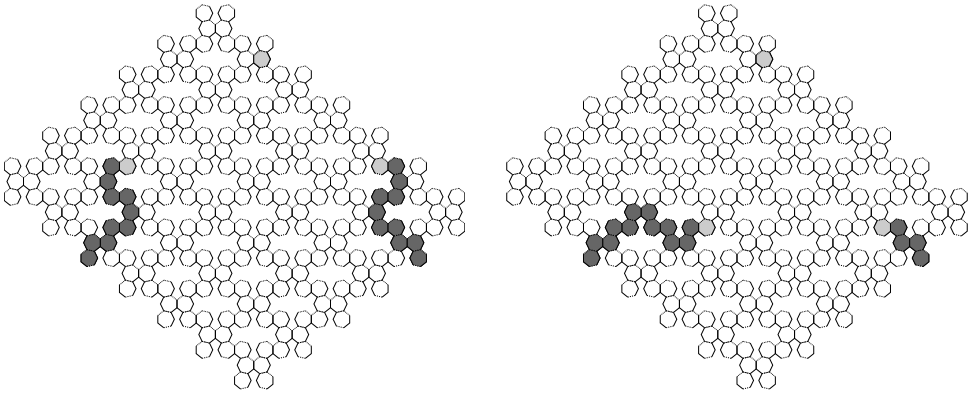
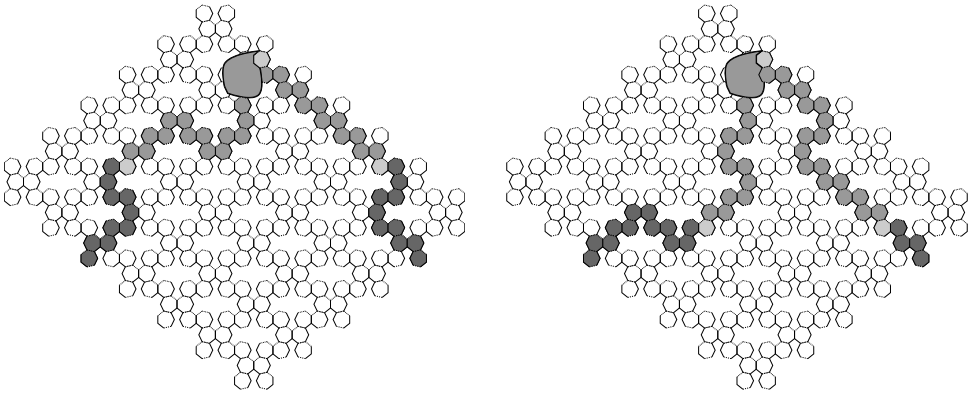


Figure 4.33: The extremal points of the bit writer configurations (lightly shaded) and the newly created extremal points (darkly shaded).



(a) (b)

Figure 4.34: Growing a path of tiles from the old extremal points to the new ones.



(a) (b)

Figure 4.35: The normalized bit writers.

4.9.4 Shifting on grid after the read

Call the last tile placed by the bit reader T_1. We now describe how the bit reader shifts back on grid after "reading a bit". This part of the construction is very similar to the construction of the on grid bit writers in Section 4.9.1. For our shifting configuration we will use the configuration obtained by removing all tiles in the bit reading configuration except for the tiles that lie on the path from the tile T_1 to the tile R (where tiles T_1 and R are as described above in Section 4.8. Without loss of generality, we assume that T_1 has standard orientation since if it is not, we can add one more tile to the path so that the last tile placed in the bit reader path is in standard orientation. We then construct an on grid bit reader in a manner similar to the way the on grid bit writers were constructed in Section 4.9.1.

4.9.5 Proof of correctness

To see that the configurations above, are indeed on grid bit reading gadgets (and thus assemblies) we make three claims: 1) every tile in the configuration completely shares an edge with another tile in the configuration and the configuration is connected, 2) the interiors of all the tiles in the configurations are pairwise disjoint, and 3) the beginning and end tiles are on grid as well as the R tile. After we see that these claims are true, then we can easily give a system which contains a bit reading gadget.

The first claim follows immediately from our construction. The construction ensured that every tile placed was next to a pre-existing tile in the assembly and in the proper orientation. The second claim also follows from the construction since we were careful to place subconfigurations sufficiently far away from each other so that there is no overlap and paths can be grown between them without overlapping.

To see claim 3, observe that the R tiles in all of the subconfigurations lie in the same position relative to some polyform on the grid (see Figure 4.31a. Consequently, once we connect the subconfigurations and shift R so that it is on grid, all of the R tiles in the sub-

configurations are on grid. Thus when the "arms" of the bit writer are grown, they start and end on grid with respect to the tile R. Hence, the beginning and end tiles are on grid as well as the R tile.

To see that we can create a system which contains a bit reading gadget using our normalized bit writers, note that we can grow a path of tiles from the last tile placed in the normalized bit writer so that it starts the growth of a bit reader at an appropriate position in relation to the bit writer. Using this notion, Figure 4.36 shows an example schematic of the complete bit reading gadget which results from reading a particular bit. The system shown can be constructed by placing appropriate glues on the tiles so that they come together as shown.

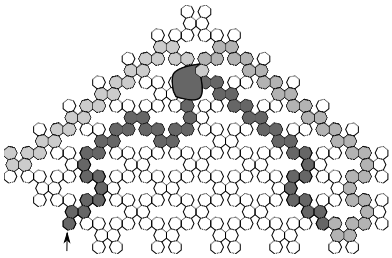


Figure 4.36: The complete bit reading gadget reading a particular bit. The arrow in this picture points to the seed of the system.

4.10 Technical appendix

In the following sections we will use this technique for computing the positions of the centers of polygonal tiles in order to show that the bit gadgets that we construct are indeed valid bit gadgets.

4.10.1 Systems with tiles shaped like a single regular polygon

In this section, we present the bit-reading gadgets for tiles shaped like a single regular polygon and the relevant calculation to show that these bit-reading gadgets are valid. Throughout this section we will use complex number to analyze configurations of polygonal

tiles. This idea is presented in Section 4.4. Many of these calculations rely on well known properties of complex numbers and regular polygons. In particular, for a complex number z, we use the equations $2\,\mathrm{Re}(z) = z + z^{-1}$ and $2\,\mathrm{Im}(z) = z - z^{-1}$. We also apply Euler's identity ($e^{i\theta} = \cos(\theta) + i\sin(\theta)$) when needed. Moreover, for a regular polygon P_n with n sides and apothem .5 (which we assume for all of the regular polygons considered here), the diameter d_n of P_n is given by the following equation which will often be used to show that two polygons do not overlap..

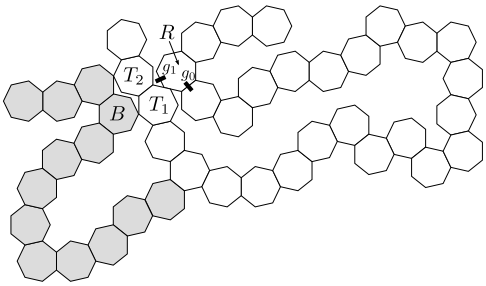
$$d_n = \frac{1}{2\cos\left(\frac{\pi}{n}\right)}$$

A bit gadget for heptagonal tiles

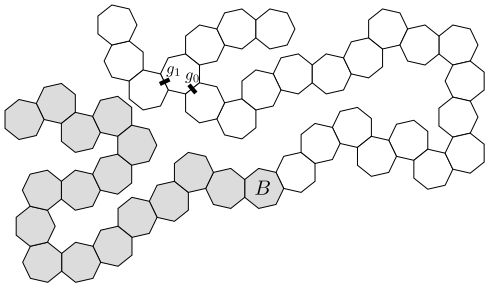
Now that we have a means of computing the exact positions of the centers of polygonal tiles, we give a bit-reader gadget that works for heptagonal tiles at temperature-1. Figure 4.37 gives a depiction of this bit-reader. Given this bit-reader, the burden of proof is two fold. 1) We must calculate the distances of the tiles in the bit-reader assembly in order to show that when a 1 is read, a 0 cannot be read and vice versa, and 2) we must show that these gadgets assemble in regular (grid-like) positions. In this section we will handle the first burden of proof, and show the second burden in Section 4.9.

In Figure 4.37, the gray tiles represent a "written" bit (either 0 or 1), while the white tiles are "reading" this bit. We ensure that the assembly sequence of a bit-gadget is such that all of the gray tiles bind before any white tiles. Referring to Figure 4.37a, we will first show that the tile labeled R in does not prevent the binding of the tile labeled T_1 or the tile labeled T_2.

Let s denote the center of the tile R, c_1 denote the center of T_1, and c_2 denote the center of T_2. This is depicted in Figure 4.38. Then, to calculate c_1 and c_2 relative to s, we assume that R is in standard orientation. Following the path of tiles lying on the dotted line in Figure 4.38 and summing the appropriate roots of unity, we obtain the polynomials $c_1 =$



(a) A 0 is read, and a 1 cannot be read by mistake since the tile B prevents a heptagonal tile from attaching via the glue labeled g_1.



(b) A 1 is read. This time a 0 cannot be read by mistake since the tile B prevents growth of a path of heptagonal tiles that attach via the glue labeled g_0. Note that some of this path may form, but B prevents the entire path from assembling, and thus prevents a 0 from being read.

Figure 4.37: A connected bit-gadget consisting of heptagonal tiles.

$\omega^6-\omega^3+\omega-\omega^4+1-\omega^4+\omega-\omega^3+1-\omega^2+\omega^5-\omega^2+\omega^4-\omega^6+\omega^4-\omega^6+\omega^4-\omega+\omega^4-1+\omega^3-\omega^6+\omega^2$

and $c_2 = c_1 - \omega^6$. By simplifying c_1, we get $c_1 = 1 + \omega - \omega^2 - \omega^3 + 2\omega^4 + \omega^5 - 2\omega^6$.

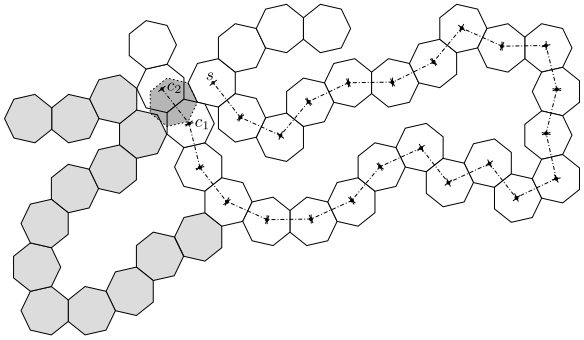


Figure 4.38: A possible configuration of the bit-reader given in Figure 4.37. We must show that the heptagonal tiles centered at c_1 and c_2 do not overlap the tile centered at s.

First, as multiplying by ω is a rotation by $2\pi/7$, it is enough to show that $\mathrm{Re}(\omega^2 c_1) \geq 1$, and to see this, consider the following.

$$\begin{aligned}
\omega^2 c_1 &= \omega^2 + \omega^3 - \omega^4 - \omega^5 + 2\omega^6 + \omega^7 - 2\omega^8 \\
&= \omega^2 + \omega^3 - \omega^4 - \omega^5 + 2\omega^6 + 1 - 2\omega \\
&= \omega^2 + \omega^3 - \omega^{-3} - \omega^{-2} + 2\omega^6 + 1 - 2\omega^{-6} \\
&= 1 + (\omega^2 - \omega^{-2}) + (\omega^3 - \omega^{-3}) + 2(\omega^6 - \omega^{-6})
\end{aligned}$$

Then since $(\omega^2 - \omega^{-2})$, $(\omega^3 - \omega^{-3})$, and $2(\omega^6 - \omega^{-6})$ are imaginary, we see $\mathrm{Re}(\omega^2 c_1) = 1$. Therefore, the heptagon with negated orientation centered at c_1 and the heptagon in standard orientation centered at s do not overlap. Note that since $\mathrm{Re}(\omega^2 c_1) = 1$, it may be that these two heptagons partially share an edge, however, the intersection of their interiors is empty.

To show that the heptagon, H_{c_2}, with standard orientation centered at c_2 and the heptagon, H_s, with negated orientation centered at s do not overlap, note that $c_2 = 1 + \omega - \omega^2 - \omega^3 + 2\omega^4 + \omega^5 - 3\omega^6$. Then, one can approximate $|c_2|$ and observe that $|c_2| > 1.11 > \frac{1}{\cos(\frac{\pi}{7})}$. Hence the distance from s to c_2 is greater than twice the diameter of one of these heptagonal tiles. Therefore, H_{c_2} and H_s do not overlap.

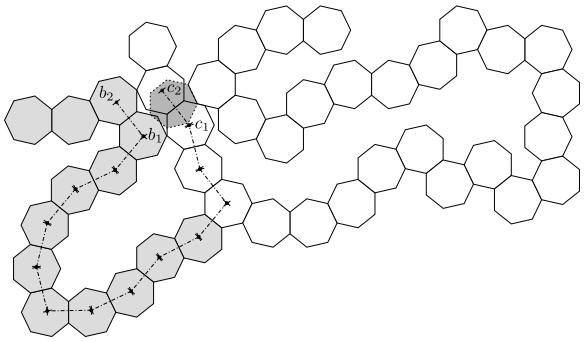


Figure 4.39: A possible configuration of the bit-reader given in Figure 4.37. We must show that the heptagonal tiles centered at c_1 and c_2 do not overlap those centered at b_1 and b_2.

Referring to Figure 4.39, relative to b_1, $c_1 = -\omega + \omega^4 - \omega + \omega^5 - \omega^2 + 1 - \omega^4 + \omega - \omega^4 + \omega - \omega^6 + \omega^2$, and $c_2 = c_1 - \omega^6$. Simplifying, $c_1 = 1 - \omega^4 + \omega^5 - \omega^6$.

To show that a heptagonal tile in negated orientation centered at b_1 and a heptagonal

tile in negated orientation centered at c_1 do not overlap, it suffices to show that $\mathrm{Re}(c_1) >= \frac{1}{2} + \frac{1}{2\cos\left(\frac{\pi}{7}\right)}$. Hence, it is enough to show that $\mathrm{Re}(c_1) = 1 - \cos\left(\frac{8\pi}{7}\right) + \cos\left(\frac{10\pi}{7}\right) - \cos\left(\frac{12\pi}{7}\right) = \frac{1}{2} + \frac{1}{2\cos\left(\frac{\pi}{7}\right)}$. Equivalently, we show $1 - 2\cos\left(\frac{8\pi}{7}\right) + 2\cos\left(\frac{10\pi}{7}\right) - 2\cos\left(\frac{12\pi}{7}\right) = \frac{1}{\cos\left(\frac{\pi}{7}\right)}$. To see this, observe

$$\begin{aligned}
2 &= -2\cos(\pi) \\
&= -\left(e^{\pi i} + e^{-\pi i}\right) \\
&= -e^{\pi i} - e^{-\pi i} + e^{\frac{\pi i}{7}} - e^{\frac{\pi i}{7}} + e^{\frac{-\pi i}{7}} - e^{\frac{-\pi i}{7}} \\
&= e^{\frac{\pi i}{7}} - e^{-\pi i} - e^{\frac{-\pi i}{7}} + e^{\frac{-\pi i}{7}} - e^{\pi i} - e^{\frac{\pi i}{7}} \\
&= e^{\frac{\pi i}{7}} - e^{-\pi i} - e^{\frac{13\pi i}{7}} + e^{\frac{-\pi i}{7}} - e^{\pi i} - e^{-13\frac{\pi i}{7}} \\
&= e^{\frac{\pi i}{7}} - e^{\frac{9\pi i}{7}} - e^{\frac{-7\pi i}{7}} + e^{\frac{11\pi i}{7}} + e^{\frac{-9\pi i}{7}} - e^{\frac{13\pi i}{7}} - e^{\frac{-11\pi i}{7}} \\
&+ e^{\frac{-\pi i}{7}} - e^{\frac{7\pi i}{7}} - e^{\frac{-9\pi i}{7}} + e^{\frac{9\pi i}{7}} + e^{\frac{-11\pi i}{7}} - e^{\frac{11\pi i}{7}} - e^{-13\frac{\pi i}{7}} \\
&= \left(e^{\frac{\pi i}{7}} + e^{\frac{-\pi i}{7}}\right)\left(1 - e^{\frac{8\pi i}{7}} - e^{\frac{-8\pi i}{7}} + e^{\frac{10\pi i}{7}} + e^{\frac{-10\pi i}{7}} - e^{\frac{12\pi i}{7}} - e^{\frac{-12\pi i}{7}}\right)
\end{aligned}$$

The last equality gives

$$1 - e^{\frac{8\pi i}{7}} - e^{\frac{-8\pi i}{7}} + e^{\frac{10\pi i}{7}} + e^{\frac{-10\pi i}{7}} - e^{\frac{12\pi i}{7}} - e^{\frac{-12\pi i}{7}} = \frac{2}{e^{\frac{\pi i}{7}} + e^{\frac{-\pi i}{7}}}$$

In other words, $1 - 2\cos\left(\frac{8\pi}{7}\right) + 2\cos\left(\frac{10\pi}{7}\right) - 2\cos\left(\frac{12\pi}{7}\right) = \frac{1}{\cos\left(\frac{\pi}{7}\right)}$, which was what we wanted. Therefore, a heptagonal tile in negated orientation and centered b_1, and a heptagonal tile in negated orientation and centered at c_1 do not overlap.

From Figure 4.39, it is now clear that a heptagonal tile in negated orientation and centered at b_1, and a heptagonal tile in standard orientation and centered at c_2 do not overlap, and that a heptagonal tile in standard orientation and centered b_2, and a heptagonal tile in standard orientation and centered at c_2 do not overlap.

Now, referring to Figure 4.40, we must show that a heptagonal tile, H_a, in negated ori-

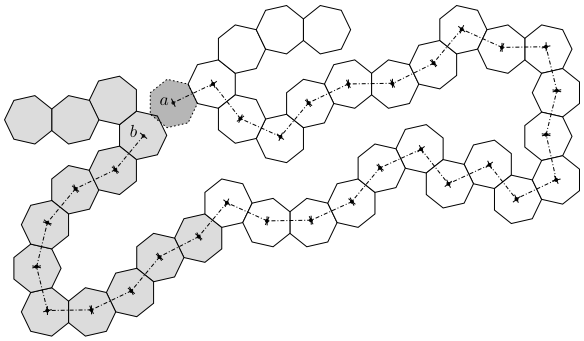
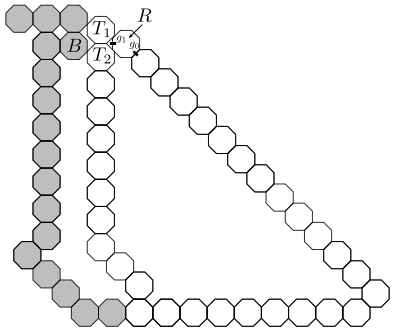


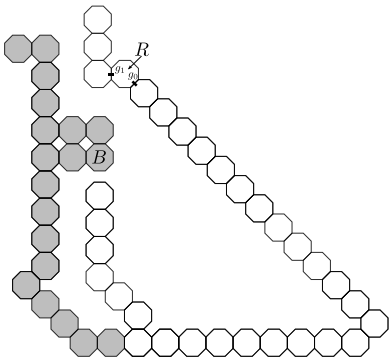
Figure 4.40: A possible configuration of the bit-reader given in Figure 4.37. We must show that the heptagonal tile centered at a overlaps the tile centered at b.

entation and centered a, and a heptagonal tile, H_b, in negated orientation and centered at b overlap. Note that relative to a, $b = -\omega^4 + \omega^6 - \omega^3 + \omega - \omega^4 + 1 - \omega^4 + \omega - \omega^3 + 1 - \omega^2 + \omega^5 - \omega^2 + \omega^4 - \omega^6 + \omega^4 - \omega^6 + \omega^4 - \omega + \omega^4 - 1 + \omega^3 - \omega + \omega^4 - \omega + \omega^4 - 1 + \omega^2 - \omega^5 + \omega - \omega^4 + \omega$. We can simplify b to obtain $b = \omega - \omega^2 - \omega^3 + 2\omega^4 - \omega^6$. Then we approximate $|b|$ to show that $|b| < 1$. Therefore H_a and H_b must overlap. Given these calculations, we can obtain a bit-reading gadget for systems whose tiles have the shape of a heptagon.

Octagonal tile assembly



(a) A 0 is read, and a 1 cannot be read by mistake since the tile B prevents a octagonal tile from attaching via the glue labeled g_1.



(b) A 1 is read. This time a 0 cannot be read by mistake since the tile B prevents growth of a path of octagonal tiles that attach via the glue labeled g_0. Note that some of this path may form, but B prevents the entire path from assembling, and thus prevents a 0 from being read.

Figure 4.41: The configurations for a bit-reading gadget consisting of octagonal tiles.

Figure 4.41 depicts two possible configurations of a bit-reading gadget construction for single-shaped systems with octagonal tiles, the gray tiles represent "bit-writer" tiles (representing either 0 or 1), while the white tiles are the "bit-reader" tiles. We ensure that the assembly sequence of a bit-gadget is such that all of the gray tiles bind before any white tiles. Referring to Figure 4.41a, we will first show that the tiles labeled R and B do not prevent the binding of the tile labeled T_1 or the tile labeled T_2. Then we will show that the tile labeled B prevents an octagonal tile from binding to the glue labeled g_1.

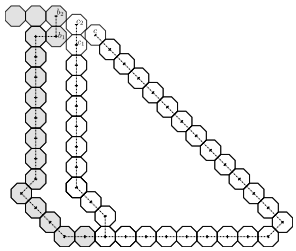


Figure 4.42: A possible configuration of the bit-reader given in Figure 4.41. We must show that the octagonal tiles centered at c_1 and c_2 do not overlap those centered at b_1 and b_2.

When analyzing even sided polygons, note that we can always assume that each polygonal tile has the standard orientation. Let T_c denote the octagonal tile centered at c and let T_{c_1} denote the octagonal tile centered at c_1 as shown in Figure 4.42. To show that T_c and T_{c_1} do not overlap, let ω now denote $e^{\frac{2\pi}{8}}$ and note that relative to c, c_1 is given by the following equation.

$$\begin{aligned} c_1 &= 13\omega^7 + \omega^5 + 8\omega^4 + \omega^2 + 2\omega^3 + 7\omega^2 \\ &= 8\omega^2 + 2\omega^3 + 8\omega^4 + \omega^5 + 13\omega^7 \end{aligned}$$

Then, after multiplying by ω we need only show that $\mathrm{Im}(\omega c_1) \leq -1$. To see this, note that $\omega c_1 = 8\omega^3 + 2\omega^4 + 8\omega^5 + \omega^6 + 13\omega^8$. Then, since $\omega^8 = 1$, $\omega^2 = i$, $\omega^4 = -1$, and $\omega^3 = \omega^{-5}$, we see that $\omega c_1 = 11 + 8(2\,\mathrm{Re}(\omega^3)) - i$, and hence, $\mathrm{Im}(\omega c_1) = -1$. Therefore, T_c and T_{c_1} do not overlap.

To show that T_c and T_{c_2} do not overlap, note that $c_2 = c_1 + \omega^2$. Then, after multiplying c_2 by ω we need only show that $\mathrm{Re}(\omega c_2) \leq -1$. To see that this inequality holds, consider the following.

$$\begin{aligned}\mathrm{Re}\,(\omega c_2) &= \mathrm{Re}\left(\omega c_1 + \omega^3\right) = \mathrm{Re}\left(11 - i + 8(2\,\mathrm{Re}(\omega^3)) + \omega^3\right) \\ &= \mathrm{Re}\left(11 + 17\,\mathrm{Re}(\omega^3)\right) = 11 - 17\frac{\sqrt{2}}{2} \\ &< 11 - 17\frac{1.414}{2} < -1\end{aligned}$$

Then, since $\mathrm{Re}(\omega c_2) \leq -1$, we see that T_c and T_{c_2} do not overlap. Therefore, T_c does not overlap T_{c_1} and T_{c_2} do not overlap. We now show that an octagonal tile, T_{c_1} say, with center c_1 and an octagonal tile, T_{b_1} say, with center b_1 do not overlap. It will then also be clear that an octagonal tile with center c_1 or c_2 and an octagonal tile with center b_1 or b_2 do not overlap. To see that T_{c_1} and T_{c_1} do not overlap, note that relative to c_1, $b_1 = 7\omega^6 + 2\omega^7 + \omega^6 + 2\omega^4 + 3\omega^3 + \omega + 7\omega^2 + 1$. It suffices to show that $\mathrm{Re}(b_1) = -1$. Then we see that $\mathrm{Re}(b_1) = 1 + \frac{\sqrt{2}}{2} - 3\frac{\sqrt{2}}{2} - 2 + 2\frac{\sqrt{2}}{2}$. Hence, $\mathrm{Re}(b_1) = -1$, and an octagonal tile with center c_1 and an octagonal tile with center b_1 do not overlap.

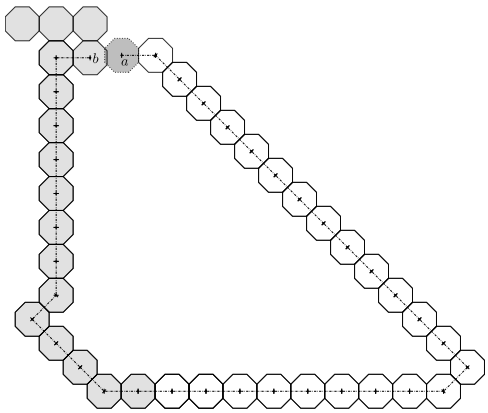


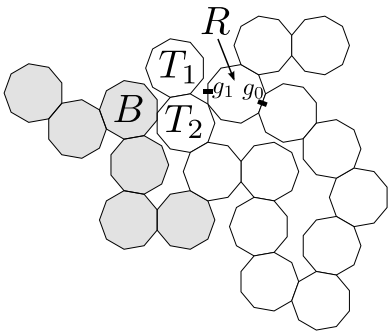
Figure 4.43: A configuration of the bit-reader given in Figure 4.41. We must show that the octagonal tile centered at a overlaps the tile centered at b.

Now, referring to Figure 4.43, in remains to be shown that an octagon with center a and an octagon with center b overlap. That is, an octagonal tile (in an existing assembly)

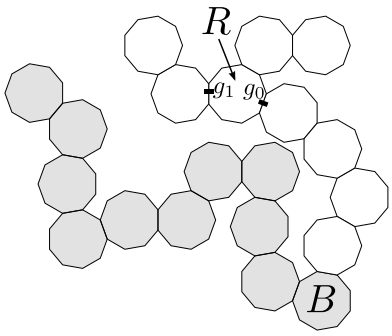
centered at b prevents the binding of an octagonal tile centered at a. To see this, note that relative to a, $b = 1 + 13\omega^7 + \omega^5 + 10\omega^4 + 3\omega^3 + \omega + 7\omega^2 + 1$. Simplifying b gives $b = 2 + \omega + 7\omega^2 + 3\omega^3 + 10\omega^4 + \omega^5 + 13\omega^7$. Then one can check that $|b| < 1$. Given these calculations, we can obtain a bit-reading gadget for systems whose tiles have the shape of a octagon.

Nonagonal tile assembly

Figure 4.44 depicts two possible configurations of a bit-reading gadget construction for single-shaped systems with nonagonal tiles, the gray tiles represent a "bit-writer" tiles (representing either 0 or 1), while the white tiles are the "bit-reader" tiles. We ensure that the assembly sequence of a bit-gadget is such that all of the gray tiles bind before any white tiles. Referring to Figure 4.44a, we will first show that the tiles labeled R and B do not prevent the binding of the tile labeled T_1 or the tile labeled T_2. Then we will show that the tile labeled B prevents an octagonal tile from binding to the glue labeled g_1.



(a) A 0 is read, and a 1 cannot be read by mistake since the tile B prevents a nonagonal tile from attaching via the glue labeled g_1.



(b) A 1 is read. This time a 0 cannot be read since the tile B prevents growth of a path of nonagonal tiles that attach via the glue labeled g_0. Note that some of this path may form, but B prevents the entire path from assembling, and thus prevents a 0 from being read.

Figure 4.44: The configurations for a bit-reading gadget consisting of nonagonal tiles.

Referring to Figure 4.45, let T_c be a nonagonal tile with negated orientation centered at c and let T_{c_1} be a nonagonal tile with negated orientation centered at c_1. We first show

that that T_c and T_{c_1} do not overlap. Let ω now denote $e^{\frac{2\pi}{9}}$.

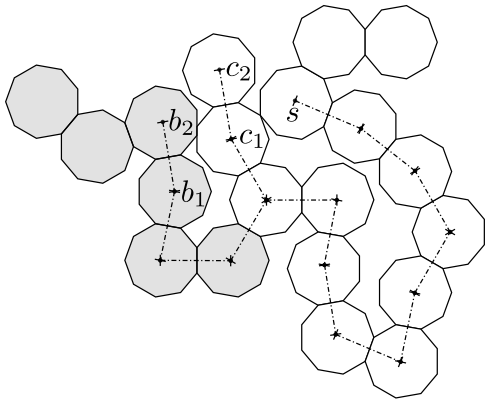


Figure 4.45: A possible configuration of the bit-reader given in Figure 4.44. We must show that the nonagonal tiles centered at c_1 and c_2 do not overlap those centered at c, b_1, and b_2.

Note that relative to c, $c_1 = -\omega^4+\omega^8-\omega^3+\omega^6-\omega^2+\omega^4-\omega^7+\omega^2-1+\omega^3$. Simplifying c_1 gives $c_1 = -1+\omega^8+\omega^6-\omega^7$. Then, after multiplying by ω^{-1} (which corresponds to rotating the Figure 4.45 clockwise by $\frac{2\pi}{9}$), it suffices to show that $\mathrm{Re}(\omega^{-1}c_1) = -\frac{1}{2} - \frac{1}{2\cos\left(\frac{\pi}{9}\right)}$. To see this, first note that $\omega^{-1}c_1 = \omega^5-\omega^6+\omega^7-\omega^8$. Therefore, $\mathrm{Re}\left(\omega^{-1}c_1\right) = \cos\left(\frac{10\pi}{9}\right)-\cos\left(\frac{12\pi}{9}\right)+\cos\left(\frac{14\pi}{9}\right)-\cos\left(\frac{16\pi}{9}\right)$. Hence, it suffices to show that

$$\cos\left(\tfrac{10\pi}{9}\right) - \cos\left(\tfrac{12\pi}{9}\right) + \cos\left(\tfrac{14\pi}{9}\right) - \cos\left(\tfrac{16\pi}{9}\right) = -\frac{1}{2} - \frac{1}{2\cos\left(\frac{\pi}{9}\right)}.$$

To see this, consider the following equations.

$$\begin{aligned}
-2 &= e^{\frac{\pi i}{9}} + e^{-\frac{\pi i}{9}} + e^{\pi i} + e^{-\pi i} - e^{-\frac{\pi i}{9}} - e^{\frac{\pi i}{9}} \\
&= e^{\frac{\pi i}{9}} + e^{-\frac{\pi i}{9}} + e^{\frac{9\pi i}{9}} + e^{-\frac{9\pi i}{9}} - e^{\frac{17\pi i}{9}} - e^{-\frac{17\pi i}{9}} \\
&= e^{\frac{\pi i}{9}} + e^{\frac{11\pi i}{9}} + e^{-\frac{9\pi i}{9}} - e^{\frac{13\pi i}{9}} - e^{-\frac{11\pi i}{9}} + e^{\frac{15\pi i}{9}} + e^{-\frac{13\pi i}{9}} - e^{\frac{17\pi i}{9}} - e^{-\frac{15\pi i}{9}} \\
&\quad + e^{-\frac{\pi i}{9}} + e^{\frac{9\pi i}{9}} + e^{-\frac{11\pi i}{9}} - e^{\frac{11\pi i}{9}} - e^{-\frac{13\pi i}{9}} + e^{\frac{13\pi i}{9}} + e^{-\frac{15\pi i}{9}} - e^{\frac{15\pi i}{9}} - e^{-\frac{17\pi i}{9}} \\
&= \left(e^{\frac{\pi i}{9}} + e^{-\frac{\pi i}{9}}\right) \\
&\quad \times \left(1 + e^{\frac{10\pi i}{9}} + e^{-\frac{10\pi i}{9}} - e^{\frac{12\pi i}{9}} - e^{-\frac{12\pi i}{9}} + e^{\frac{14\pi i}{9}} + e^{-\frac{14\pi i}{9}} - e^{\frac{16\pi i}{9}} - e^{-\frac{16\pi i}{9}}\right)
\end{aligned}$$

Therefore, $-\frac{2}{e^{\pi i/9}+e^{-\pi i/9}} = 1 + e^{10\pi i/9} + e^{-10\pi i/9} - e^{12\pi i/9} - e^{-12\pi i/9} + e^{14\pi i/9} + e^{-14\pi i/9} - e^{16\pi i/9} - e^{-16\pi i/9}$. Using the identity $\cos(\theta) = \frac{e^{i\theta}+e^{-i\theta}}{2}$, we can see that $-1 - \frac{1}{\cos\left(\frac{\pi i}{9}\right)} = 2\cos\left(\frac{10\pi i}{9}\right) - 2\cos\left(\frac{12\pi i}{9}\right) + 2\cos\left(\frac{14\pi i}{9}\right) - 2\cos\left(\frac{16\pi i}{9}\right)$. Therefore, T_c and T_{c_1} do not overlap.

Now we let T_{b_1} denote a nonagonal tile with negated orientation centered at b_1 and show that T_{b_1} and T_{c_1} do not overlap. Relative to c_1, $b_1 = -\omega^3 + \omega^6 - 1 + \omega^2$. It suffices to show that $\mathrm{Re}(\omega^{-1}b_1) < -\frac{1}{2} - \frac{1}{2\cos(\pi/9)}$, which we can numerically verify is true by approximating each side of the inequality.

Similarly, we let T_{b_2} denote a nonagonal tile with standard orientation centered at b_2 and show that T_{b_2} and T_{c_1} do not overlap. Relative to c_1, $b_2 = -\omega^3 + \omega^6 - 1 + \omega^2 - \omega^7$. Then, it suffices to show that $\mathrm{Re}(b_2) = -1$. To see this, note that $b_2 = -1 - (\omega^3 - \omega^{-3}) + (\omega^2 - \omega^{-2})$. Since $\omega^3 - \omega^{-3}$ and $\omega^2 - \omega^{-2}$ are imaginary, $\mathrm{Re}(b_2) = -1$. Therefore, T_{c_1} and T_{b_2} do not overlap. Similarly, we can see that a nonagonal tile centered at c_2 that is in standard orientation and a nonagonal tile centered at b_2 that is in standard orientation do not overlap.

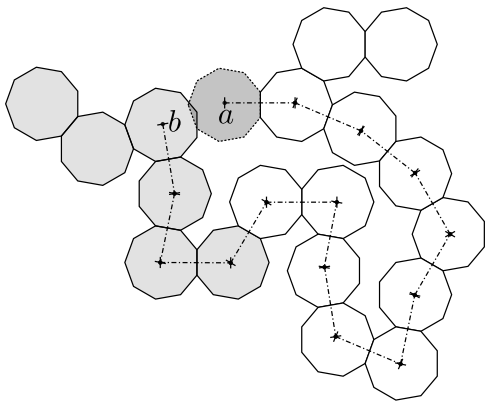


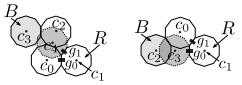
Figure 4.46: A configuration of the bit-reader given in Figure 4.44. We must show that the nonagonal tile centered at a overlaps the tile centered at b.

Now, referring to Figure 4.46, it remains to be shown that a nonagonal tile, which we will denote by T_a, centered at a that is in standard orientation and a nonagonal tile, which we will denote by T_b, centered at b that is in standard orientation overlap. Relative to a, $b = 1 - \omega^4 + \omega^8 - \omega^3 + \omega^6 - \omega^2 + \omega^4 - \omega^7 + \omega^2 - 1 + \omega^6 - 1 + \omega^2 - \omega^7$. Simplifying b gives $b = -1 + \omega^2 - \omega^3 + 2\omega^6 - 2\omega^7 + \omega^8$. Then we can approximate $|b|$ to see that $|b| < 1$.

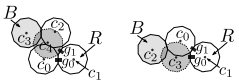
Therefore, T_a and T_b overlap.

Polygonal tile assembly with 10, 11, or 12 sided regular polygonal tiles

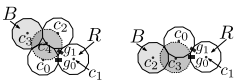
In the cases where tiles consist of regular polygons with $10, 11$, or 12 sides, bit-reading gadgets are relatively simple to construct. Figure 4.47 depicts the bit-reading gadgets for each case. Note that since each polygonal tile of these bit-reading gadgets abuts another tile, we need only show that for each configuration depicted in Figure 4.47, of the two exposed glues, g_0 and g_1 of the tile R, a tile can only attach to one of these glues depending on the position of the tile B in the figure. In other words, for each configuration depicted in Figure 4.47, we show that the intersection of the interiors of a polygon with the same shape, position and orientation as B and a polygon with the same shape, position and orientation of the gray tile's position and orientation.



(a) Bit-reading gadget configuration for decagonal tiles.



(b) Bit-reading gadget configuration for hendecagonal tiles.



(c) Bit-reading gadget configuration for dodecagonal tiles.

Figure 4.47: (a), (b) and (c) each depict two configurations of polygonal tiles which represents either a 0 (bottom) or a 1 (top).

For decagonal tiles, let $\omega = e^{\frac{2\pi i}{10}}$ and consider Figure 4.47a. To show that this gives a valid bit-reader, we first show that using the top assembly depicted in the top figure of Figure 4.47a, a polygon centered at c_2 and a polygon centered at c_3 overlap. Note that relative to c_1, $c_3 = -1$ and $c_2 = \omega^4 + \omega^6$. Hence, $c_2 = \omega^4 + \omega^{-4} = 2\,\mathrm{Re}\,(\omega^4) = 2\cos\left(\frac{8\pi}{10}\right)$. Then the distance d from c_3 to c_2 satisfies $d = |-1-2\cos\left(\frac{8\pi}{10}\right)| < .62$.

Secondly, we show that in the bottom figure of Figure 4.47a, a polygon centered at c_3 and a polygon centered at c_4 overlap. Relative to c_4, $c_3 = \omega^9 - 1 + \omega^2 - 1$. Hence, $c_3 = -2 + \omega^2 + \omega^9$. Then, $|c_3| = \left(-2 + \cos\left(\frac{4\pi}{10}\right) + \cos\left(\frac{18\pi}{10}\right)\right)^2 + \left(\sin\left(\frac{4\pi}{10}\right) + \sin\left(\frac{18\pi}{10}\right)\right)^2 < .91$.

For hendecagonal tiles, let $\omega = e^{\frac{2\pi i}{11}}$ and consider Figure 4.47b. To show that this gives a valid bit-reader, we first show that using the top assembly depicted in the top fig-

ure of Figure 4.47b, a polygon centered at c_2 and a polygon centered at c_3 overlap. Note that relative to c_3, $c_2 = 1 - \omega^{10} + \omega^6$. Hence, $|c_2|^2 = \left(1 - \cos\left(\frac{20\pi}{11}\right) + \cos\left(\frac{12\pi}{11}\right)\right)^2 + \left(-\sin\left(\frac{20\pi}{11}\right) + \sin\left(\frac{12\pi}{11}\right)\right)^2 < .71$

Secondly, we show that in the bottom figure of Figure 4.47b, a polygon centered at c_2 and a polygon centered at c_1 do not overlap. Relative to c_1, $c_2 = -1 + \omega^2$. Then, $|c_2|^2 = \left(-1 + \cos\left(\frac{4\pi}{11}\right)\right)^2 + \sin^2\left(\frac{4\pi}{11}\right) > \frac{1}{\cos\left(\frac{\pi}{11}\right)}$.

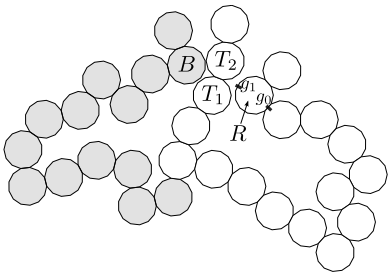
For dodecagonal tiles, let $\omega = e^{\frac{2\pi i}{12}}$ and consider Figure 4.47c. To show that this gives a valid bit-reader, we first show that using the top assembly depicted in the top figure of Figure 4.47c, a polygon centered at c_2 and a polygon centered at c_3 overlap. Note that relative to c_3, $c_2 = 1 + \omega^5 + \omega^7$. Hence, $|c_2|^2 = \left(1 + \cos\left(\frac{10\pi}{12}\right) + \cos\left(\frac{14\pi}{12}\right)\right)^2 + \left(\sin\left(\frac{10\pi}{12}\right) + \sin\left(\frac{14\pi}{12}\right)\right)^2 < .54$

Secondly, we show that in the bottom figure of Figure 4.47c, a polygon centered at c_2 and a polygon centered at c_1 do not overlap. Relative to c_1, $c_2 = -1 + \omega^2$. Then, it suffices to show that $\omega^2 c_2 = -1$. Note that

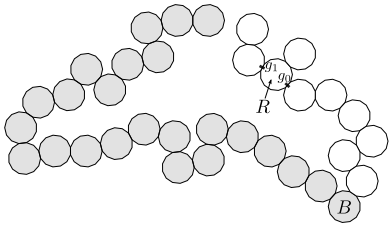
$$\begin{aligned}\omega^2 c_2 &= -\omega^2 + \omega^4 = \omega^{-4} + \omega^4 \\ &= 2\,\mathrm{Re}\left(\omega^4\right) = 2\cos\left(\tfrac{8\pi}{12}\right) \\ &= 2\cos\left(\tfrac{2\pi}{3}\right) = -1\end{aligned}$$

Tridecagonal tile assembly

Figure 4.48 depicts two possible configurations of a bit-reading gadget construction for single-shaped systems with tridecagonal tiles, the gray tiles represent a "bit-writer" tiles (representing either 0 or 1), while the white tiles are the "bit-reader" tiles. We ensure that the assembly sequence of a bit-gadget is such that all of the gray tiles bind before any white tiles. Referring to Figure 4.48a, we will first show that the tiles labeled R and B do not prevent the binding of the tile labeled T_1 or the tile labeled T_2. Then we will show that



(a) A 0 is read, and a 1 cannot be read by mistake since the tile B prevents a tridecagonal tile from attaching via the glue labeled g_1.



(b) A 1 is read. This time a 0 cannot be read by mistake since the tile B prevents growth of a path of tridecagonal tiles that attach via the glue labeled g_0. Note that some of this path may form, but B prevents the entire path from assembling, and thus prevents a 0 from being read.

Figure 4.48: The configurations for a bit-reading gadget consisting of tridecagonal tiles.

the tile labeled B prevents an octagonal tile from binding to the glue labeled g_1.

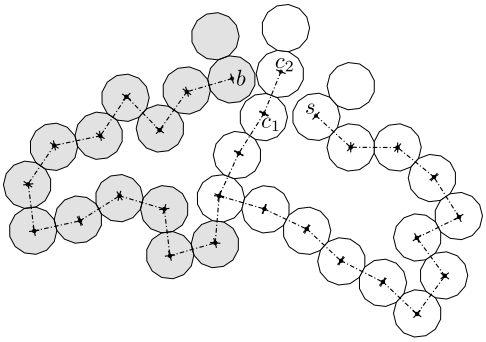


Figure 4.49: A possible configuration of the bit-reader given in Figure 4.48. We must show that the nonagonal tiles centered at c_1 and c_2 do not overlap those centered at c and b.

We now refer to Figure 4.49 and let ω be $e^{\frac{2\pi i}{13}}$. Let T_c denote the tridecagonal tile with negated orientation centered at c and let T_{c_1} denote the tridecagonal tile with negated orientation centered at c_1. To show that T_c and T_{c_1} do not overlap, note that relative to c, c_1 is given by $c_1 = -\omega^5 + 1 - \omega^5 + \omega^{11} - \omega^1 + \omega^{11} - \omega^2 + \omega^5 - \omega^{12} + \omega^5 - \omega^{12} + \omega^6 - \omega^9 + \omega^2$ and $c_2 = c_1 - \omega^9$. Simplifying c_1, we obtain $c_1 = -2\omega^{12} + 2\omega^{11} - \omega^9 + \omega^6 - \omega + 1$. Then by approximating $|c_1|$ we can see that $|c_1| > 1.13 > \frac{1}{\cos\left(\frac{\pi}{13}\right)}$. Therefore, T_c and T_{c_1} do not overlap.

Now let T_{c_2} denote the tridecagonal tile with standard orientation centered at c_2. Since $c_2 = c_1 - \omega^9$, we see that $c_2 = -2\omega^{12} + 2\omega^{11} - 2\omega^9 + \omega^6 - \omega + 1$. Then we approximate $|c_2|$

to show that $|c_2| > 1.21 > \frac{1}{\cos\left(\frac{\pi}{13}\right)}$. Therefore, T_c and T_{c_2} do not overlap.

Let T_b denote the tridecagonal tile with standard orientation centered at b. Then, relative to b, $c_1 = \omega^7 - \omega^2 + \omega^5 - \omega^2 + \omega^7 - \omega^2 + \omega^{10} - \omega^7 + \omega - \omega^6 + \omega^{10} - \omega^7 + \omega^3 - \omega^9 + \omega^2$. We can simplify c_1 to obtain $c_1 = 2\omega^{10} - \omega^9 - \omega^6 + \omega^5 + \omega^3 - 2\omega^2 + \omega$. Also note that $c_2 = c_1 - \omega^9$.

Then, we approximate $|c_1|$ to show that $|c_1| > 1.06 > \frac{1}{\cos\left(\frac{\pi}{13}\right)}$. Therefore, T_b and T_{c_1} do not overlap. Similarly, $|c_2| > 1.04 > \frac{1}{2} + \frac{1}{2\cos(\pi/13)}$, and so T_b and T_{c_1} do not overlap.

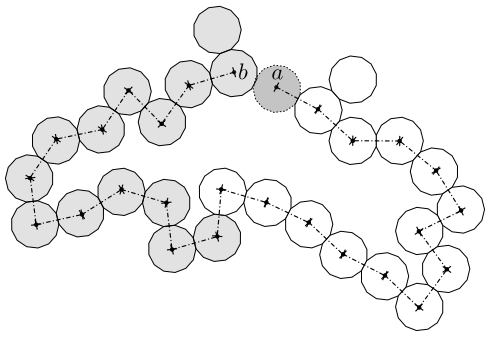
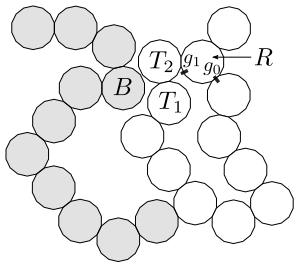


Figure 4.50: A configuration of the bit-reader given in Figure 4.48. We must show that the tridecagonal tile centered at a overlaps the tile centered at b.

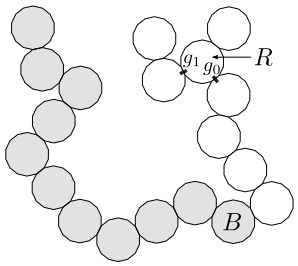
Now, referring to Figure 4.50, it remains to be shown that a tridecagonal tile, which we will denote by T_a, centered at a that is in standard orientation and a tridecagonal tile, which we will denote by T_b, centered at b that is in standard orientation overlap. Relative to b,

$$\begin{aligned} a &= \omega^7 - \omega^2 + \omega^5 - \omega^2 + \omega^7 - \omega^2 + \omega^{10} - \omega^7 + \omega - \omega^6 + \omega^{10} - \omega^7 + \omega^3 - \omega^6 + \omega^{12} - \omega^5 \\ &\quad + \omega^{12} - \omega^5 + \omega^2 - \omega^{11} + \omega - \omega^{11} + \omega^5 - 1 + \omega^5 - \omega^{12} \\ &= -1 + 2\omega - 2\omega^2 + \omega^3 + \omega^5 - 2\omega^6 + 2\omega^{10} - 2\omega^{11} + \omega^{12} \end{aligned}$$

Then we can approximate $|a|$ to see that $|a| < 1$. Therefore, T_a and T_b overlap.



(a) A 0 is read, and a 1 cannot be read by mistake since the tile B prevents a tetradecagonal tile from attaching via the glue labeled g_1.



(b) A 1 is read. This time a 0 cannot be read by mistake since the tile B prevents growth of a path of tetradecagonal tiles that attach via the glue labeled g_0. Note that some of this path may form, but B prevents the entire path from assembling, and thus prevents a 0 from being read.

Figure 4.51: The configurations for a bit-reading gadget consisting of tetradecagonal tiles.

Tetradecagonal tile assembly

Figure 4.51 depicts two possible configurations of a bit-reading gadget construction for single-shaped systems with tetradecagonal tiles, the gray tiles represent a "bit-writer" tiles (representing either 0 or 1), while the white tiles are the "bit-reader" tiles. We ensure that the assembly sequence of a bit-gadget is such that all of the gray tiles bind before any white tiles. Referring to Figure 4.51a, we will first show that the tiles labeled R and B do not prevent the binding of the tile labeled T_1 or the tile labeled T_2. Then we will show that the tile labeled B prevents an octagonal tile from binding to the glue labeled g_1.

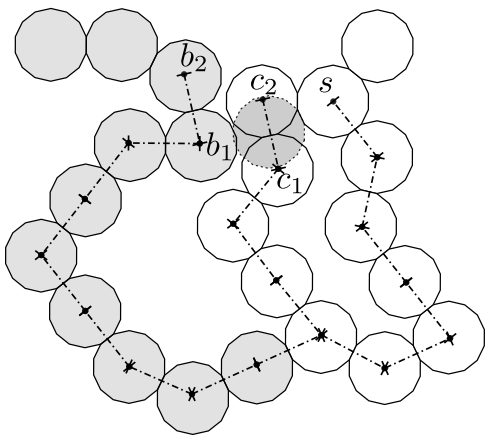


Figure 4.52: A possible configuration of the bit-reader given in Figure 4.51. We must show that the nonagonal tiles centered at c_1 and c_2 do not overlap those centered at c, b_1 and b_2.

We now refer to Figure 4.52 and let ω be $e^{\frac{2\pi i}{13}}$. Let T_c, T_{c_1}, T_{c_2}, T_{b_1}, and T_{b_2} denote the tridecagonal tile with standard orientation centered at c, c_1, c_2, b_1, and b_2 respectively. Then, to show that T_c and T_{c_1} do not overlap, note that relative to c, c_1 is given by $c_1 = c_1 = 3\omega^{12} + \omega^{10} + \omega^8 + \omega^6 + 2\omega^5 + \omega^2$. Then by approximating $|c_1|$ we can see that $|c_1| > 1.2 > \frac{1}{\cos\left(\frac{\pi}{14}\right)}$. Therefore, T_c and T_{c_1} do not overlap. Moreover, $c_2 = c_1 + \omega^4$. Then, consider the following.

$$\begin{aligned} c_2 &= 3\omega^{12} + \omega^{10} + \omega^8 + \omega^6 + 2\omega^5 + \omega^4 + \omega^2 \\ &= (2\omega^{12} + 2\omega^5) + (\omega^{12} + \omega^{10} + \omega^8 + \omega^6 + \omega^4 + \omega^2) \qquad (4.1) \\ &= (2\omega^{12} - 2\omega^{12}) + (\omega^{12} + \omega^{10} + \omega^8 + \omega^6 + \omega^4 + \omega^2 + 1 - 1) \qquad (4.2) \\ &= -1 \end{aligned}$$

Equation (4.1) follows from the following equalities that

$$\omega^5 = e^{\left(\frac{10\pi i}{14}\right)} = -e^{\left(\frac{10\pi i}{14} + \frac{14\pi i}{14}\right)} = -e^{\left(\frac{24\pi i}{14}\right)} = -\omega^{12}.$$

Equation (4.2) follows from the fact that $\omega^{12} + \omega^{10} + \omega^8 + \omega^6 + \omega^4 + \omega^2 + 1 = 0$. To see this, note that $\omega^{12} + \omega^{10} + \omega^8 + \omega^6 + \omega^4 + \omega^2 + 1 = \omega^2\left(\omega^{12} + \omega^{10} + \omega^8 + \omega^6 + \omega^4 + \omega^2 + 1\right)$, and so,

$$\left(\omega^2 - 1\right)\left(\omega^{12} + \omega^{10} + \omega^8 + \omega^6 + \omega^4 + \omega^2 + 1\right) = 0.$$

Then, since $\omega^2 - 1 \neq 0$, it follows that $\omega^{12} + \omega^{10} + \omega^8 + \omega^6 + \omega^4 + \omega^2 + 1 = 0$. Therefore, T_c and T_{c_2} do not overlap.

Now, to show that T_{b_1} does not overlap T_{c_1} or T_{c_2}, note that relative to b_1, $c_1 = -1 + 2\omega^9 + 2\omega^{12} + \omega^{13} + 2\omega + 2\omega^5 + \omega^2$. Simplifying c_1, we obtain $c_1 = -1 + \omega^9 + \omega^{13} + 2\omega$. Then we can approximate $|c_1|$ to see that $|c_1| > 1.1 > \frac{1}{\cos(\pi/14)}$. Similarly, relative to b_1, $|c_2| > 1.06 > \frac{1}{\cos(\pi/14)}$. Therefore, T_{b_1} does not overlap T_{c_1} or T_{c_2}. This also shows that T_{b_2}

and T_{c_2} do not overlap since relative to b_2, $c_2 = -\omega^4 + c_1 + \omega^4$.

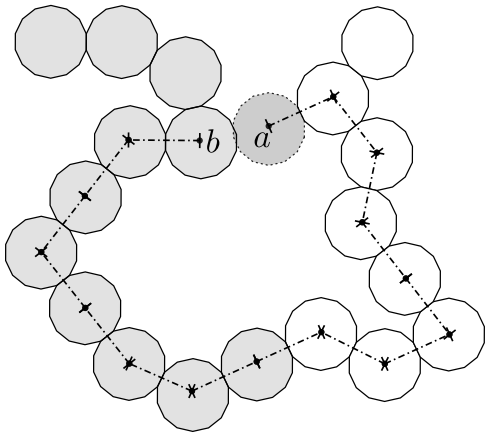
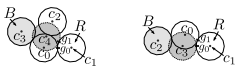


Figure 4.53: A configuration of the bit-reader given in Figure 4.51. We must show that the tetradecagonal tile centered at a overlaps the tile centered at b.

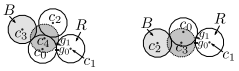
Now, referring to Figure 4.53, it remains to be shown that a tetradecagonal tile, which we will denote by T_a, centered at a that is in standard orientation and a tetradecagonal tile, which we will denote by T_b, centered at b that is in standard orientation overlap. Relative to b, $a = -1 + 2\omega^9 + 2\omega^{12} + \omega^{13} + 2\omega + \omega^{13} + \omega + 2\omega^5 + \omega^3 + \omega^5 + \omega^8$. Simplifying a, we see $a = -1 + 3\omega + \omega^3 + \omega^5 + \omega^8 + 2\omega^9 + 2\omega^{13}$. Then we can approximate $|a|$ to see that $|a| < 1$. Therefore, T_a and T_b overlap.

Polygonal tile assembly with regular polygonal tiles with 15 or more sides

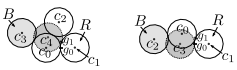
In the cases where tiles consist of regular polygons with 15 or more sides, we give a general scheme for obtaining bit-reading gadgets for each case. Figure 4.54 depicts the bit-reading gadgets for each case. Note that since each polygonal tile of these bit-reading gadgets abuts another tile, we need only show that for each configuration depicted in Figure 4.54, of the two exposed glues, g_0 and g_1 of the tile R, a tile can only attach to one of these glues depending on the position of the tile B in the figure. In other words, for each configuration depicted in Figure 4.54, we show that the intersection of the interiors of a polygon with the same shape, position and orientation as B and a polygon with the same shape, position and orientation of the gray tile's position and orientation.



(a) Bit-reading gadget configuration for pentadecagonal tiles.



(b) Bit-reading gadget configuration for hexadecagonal tiles.



(c) Bit-reading gadget configuration for heptadecagonal tiles.

Figure 4.54: (a), (b) and (c) each depict two configurations of polygonal tiles which represents either a 0 (bottom) or a 1 (top).

Now, consider a polygon P_n with $n \geq 15$ sides and let ω be the n^{th} root of unity $e^{\frac{2\pi i}{n}}$. Then, the general scheme for constructing a bit-reading gadget falls into two cases. First, if n is odd (the cases where n is even are similar), relative to a tile with negated orientation (the polygon labeled R in the configurations in Figure 4.54), the two configurations that give rise to the bit-reading gadget are as follows. Let k be such that $n = 2k + 1$ ($n = 2k$ if n is even). To "read" a 1, the configuration is obtained by centering a blocker tile with negated orientation, labeled B in the top configurations of Figure 4.54, at $-\omega^{n-1} + \omega^{k+1}$ (whether n is even or odd). Then R exposes two glues g_1 and g_0 such that if a tile binds to g_1, it will have standard orientation and be centered at $-\omega^{n-1}$ (whether n is even or odd) and if a tile that binds to g_0, it will have standard orientation and be centered at -1. We will show that B will prevent this tile from binding. This gives the configuration depicted in the top figures of Figure 4.54. Similarly, to "read" a 0, the configuration is obtained by centering a blocker tile with negated orientation, labeled B in the bottom configuration of Figure 4.54a, at $-1 + \omega^{k-1}$ ($-1 + \omega^{k-2}$ if n is even) relative to R. In this case, we will show that B prevents a tile from binding to g_1. In addition, we place a glue on the tile that binds to g_0 that allows for another tile to bind to it so that its center is at $c_2 = -1 + \omega^{\lfloor \frac{k-1}{2} \rfloor}$ ($c_2 = -1 + \omega^{\frac{k-2}{2}}$ if n is even). This gives the configuration depicted in the bottom figures of Figure 4.54a and Figure 4.54c. Moreover, we show that neither R nor B prevent the binding of this tile.

In order to perform the calculations used to show the correctness of these bit-reading gadgets, we consider the cases where n is even and where n is odd.

Case 1: (n is odd)

Suppose that $n = 2k + 1$ for some k. To show that a polygon centered at c_1 and a polygon centered at c_2 do not overlap, consider the case where k is odd. Note that relative to c_0, $c_1 = 1$ and $c_2 = \omega^{\frac{k-1}{2}}$. Then the distance d_n from c_1 to c_2 satisfies the following equation.

$$d_n^2 = \left(1 - \cos\left(\frac{(k-1)\,\pi}{n}\right)\right)^2 + \sin^2\left(\frac{(k-1)\,\pi}{n}\right)$$

Substituting $k = \frac{n-1}{2}$ for k and simplifying, we obtain $d_n^2 = 2 + 2\sin\left(\frac{3\pi}{2n}\right)$. Now to show that a polygon centered at c_1 and a polygon centered at c_2 do not overlap, we show that $d_n^2 > \frac{1}{\cos^2\left(\frac{\pi}{n}\right)}$ for $n \geq 15$. To see this, note that $\cos^2\left(\frac{\pi}{n}\right) d_n^2 = 2\cos^2\left(\frac{\pi}{n}\right)\left(1 + \sin\left(\frac{3\pi}{2n}\right)\right)$. Then for $n \geq 15$, $2\cos^2\left(\frac{\pi}{n}\right)\left(1 + \sin\left(\frac{3\pi}{2n}\right)\right) > 2\cos^2\left(\frac{\pi}{4}\right) = 1$.

It then follows that $d_n > \frac{1}{\cos\left(\frac{\pi}{n}\right)}$, and therefore d_n is greater than twice the circumradius of our polygons. Hence, a polygon centered at c_1 and a polygon centered at c_2 do not overlap.

In the case where k is even, let m be such that $k = 2m$. Then relative to c_0, $c_1 = 1$ and $c_2 = \omega^{m-1}$. In this case, d_n satisfies the following equation.

$$d_n^2 = \left(1 - \cos\left(\frac{(2m-2)\,\pi}{n}\right)\right)^2 + \sin^2\left(\frac{(2m-2)\,\pi}{n}\right)$$

Substituting $m = \frac{k}{2}$ for m and $k = \frac{n-1}{2}$ for k we obtain $d_n^2 = 2 + 2\sin\left(\frac{5\pi}{2n}\right)$. Now to show that a polygon centered at c_1 and a polygon centered at c_2 do not overlap, we show that $d_n^2 > \frac{1}{\cos^2\left(\frac{\pi}{n}\right)}$ for $n \geq 15$. To see this, note that $\cos^2\left(\frac{\pi}{n}\right) d_n^2 = 2\cos^2\left(\frac{\pi}{n}\right)\left(1 + \sin\left(\frac{5\pi}{2n}\right)\right)$. Then for $n \geq 15$, $2\cos^2\left(\frac{\pi}{n}\right)\left(1 + \sin\left(\frac{5\pi}{2n}\right)\right) > 2\cos^2\left(\frac{\pi}{4}\right) = 1$.

It then follows in the case where k is even, $d_n > \frac{1}{\cos\left(\frac{\pi}{n}\right)}$, and therefore d_n is greater than twice the circumradius of our polygons. Hence, a polygon centered at c_1 and a polygon centered at c_2 do not overlap.

Now, to show that a polygon centered at c_3 and a polygon centered at c_4 overlap, note that relative to c_1, $c_3 = -1 + \omega^{k-1}$ and $c_4 = -\omega^{n-1}$. Therefore, the distance d_n from c_3 to

c_4 is satisfies the following equation.

$$d_n^2 = \left(-1 + \cos\left(\frac{2(k-1)\pi}{n}\right) + \cos\left(\frac{2(n-1)\pi}{n}\right)\right)^2 + \left(\sin\left(\frac{2(k-1)\pi}{n}\right) + \sin\left(\frac{2(n-1)\pi}{n}\right)\right)^2$$

Substituting $k = \frac{n-1}{2}$ for k and simplifying, we obtain,

$$d_n^2 = 1 + 2\left(2\sin^2\left(\frac{\pi}{n}\right)\left(1 - 2\cos\left(\frac{\pi}{n}\right)\right)\right)$$

. Note that for each $n > 2$, $d_n^2 < 1$. To see this, it suffices to show that

$$2\sin^2\left(\frac{\pi}{n}\right)\left(1 - 2\cos\left(\frac{\pi}{n}\right)\right) < 0$$

. This follows from the fact that $2\sin^2\left(\frac{\pi}{n}\right) > 0$ and $1 - 2\cos\left(\frac{\pi}{n}\right) < 0$ for $n > 2$.

Now, since for each $n > 2$, $d_n^2 < 1$, we see that $d_n < 1$. Since the length of the apothem for each tile is assumed to be $\frac{1}{2}$, we can conclude that a polygon centered at c_3 and a polygon centered at c_4 must overlap.

Case 2: (n is even)

Let k be such that $n = 2k$. Then, relative to c_0, $c_1 = 1$ and $c_2 = \omega^{\lfloor \frac{k-2}{2} \rfloor}$. Then the distance, d_n say, from c_1 to c_2 satisfies the following equation

$$d_n = \left(1 - \cos\left(\frac{(k-2)\,\pi}{n}\right)\right)^2 + \sin^2\left(\frac{(k-2)\,\pi}{n}\right)$$

.

Substituting $k = \frac{n}{2}$ for k and simplifying, we obtain $d_n^2 = 2 + 2\sin\left(\frac{2\pi}{n}\right)$. To show that c_0 and c_1 do not overlap, it suffices to show that $d_n^2 > \frac{1}{\cos^2\left(\frac{\pi}{n}\right)}$. To see this, note that $\cos^2\left(\frac{\pi}{n}\right) d_n^2 = 2\cos^2\left(\frac{\pi}{n}\right)\left(1 + \sin\left(\frac{2\pi}{n}\right)\right)$. Then for $n \geq 16$, $2\cos^2\left(\frac{\pi}{n}\right)\left(1 + \sin\left(\frac{2\pi}{n}\right)\right) >$

$2\cos^2\left(\frac{\pi}{4}\right) = 1$.

As in the case where n is odd, it then follows that in the case where n is even, $d_n > \frac{1}{\cos\left(\frac{\pi}{n}\right)}$, and therefore d_n is greater than twice the circumradius of our polygons. Hence, a polygon centered at c_1 and a polygon centered at c_2 do not overlap.

Now, to show that a polygon centered at c_3 and a polygon centered at c_4 overlap, note that relative to c_1, $c_3 = -1 + \omega^{k-2}$ and $c_4 = -\omega^{n-1}$. Therefore, the distance d_n from c_3 to c_4 is satisfies the following equation.

$$d_n^2 = \left(-1 + \cos\left(\frac{2(k-2)\pi}{n}\right) + \cos\left(\frac{2(n-1)\pi}{n}\right)\right)^2 + \left(\sin\left(\frac{2(k-2)\pi}{n}\right) + \sin\left(\frac{2(n-1)\pi}{n}\right)\right)^2$$

Substituting $k = \frac{n}{2}$ for k and simplifying, we obtain, $d_n^2 = 1 - 8\left(\sin^2\left(\frac{\pi}{n}\right)\cos\left(\frac{2\pi}{n}\right)\right)$. Note that for each $n \geq 16$, $d_n^2 < 1$. To see this, it suffices to show that $-8\left(\sin^2\left(\frac{\pi}{n}\right)\cos\left(\frac{2\pi}{n}\right)\right) < 0$. This follows from the fact that $8\sin^2\left(\frac{\pi}{n}\right) > 0$ and $\cos\left(\frac{2\pi}{n}\right) > 0$ for $n > 16$.

Now, since for each $n \geq 16$, $d_n^2 < 1$, we see that $d_n < 1$. Since the length of the apothem for each tile is assumed to be $\frac{1}{2}$, we can conclude that a polygon centered at c_3 and a polygon centered at c_4 must overlap.

4.10.2 2-shaped systems with regular polygonal tiles

The following figures give configurations for normalized on-grid bit-reading gadgets that can be used to obtain bit-reading assemblies for 2-shaped systems where the tiles of the system have the shape of two different regular polygons. Note that the grid construction techniques from Section 4.5 can be used to obtain the grids shown using dashed lines in the figures below.

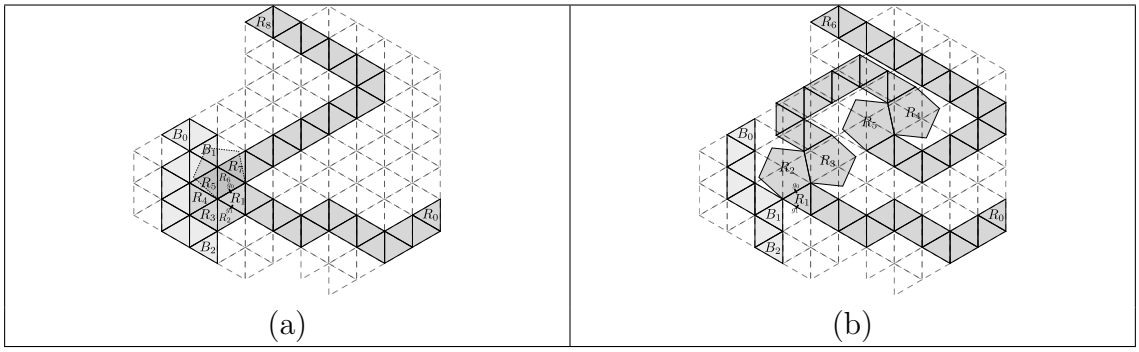


Table 4.1: Configurations for normalized on-grid bit-reading gadgets that can be used for 2-shaped systems using whose tiles have the shape of a regular triangle and a regular pentagon. (a) represents a 0, and (b) represents a 1.

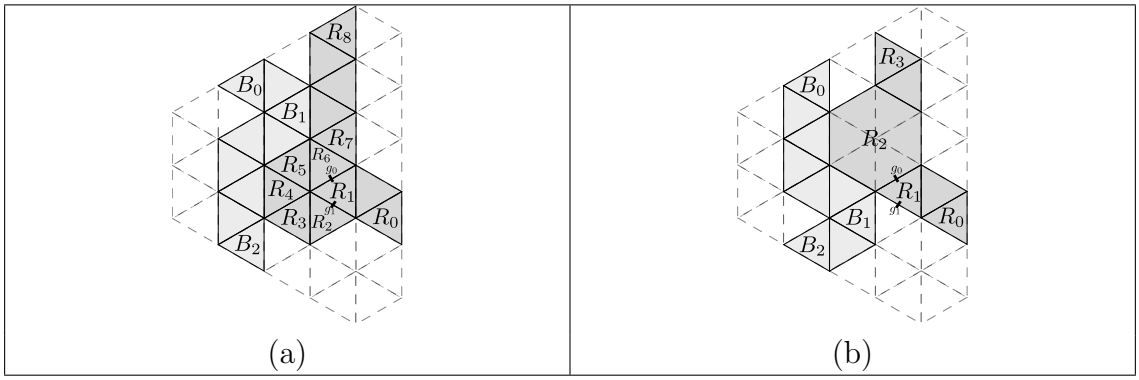


Table 4.2: Configurations for normalized on-grid bit-reading gadgets that can be used for 2-shaped systems using whose tiles have the shape of a regular triangle and a regular hexagon. (a) represents a 0, and (b) represents a 1.

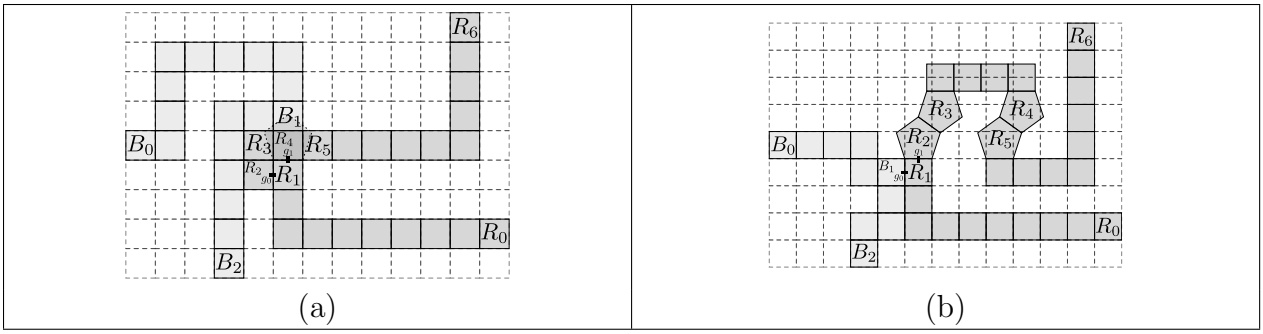


Table 4.3: Configurations of for normalized on-grid bit-reading gadgets that can be used for 2-shaped systems using whose tiles have the shape of a square and a regular pentagon. (a) represents a 0, and (b) represents a 1.

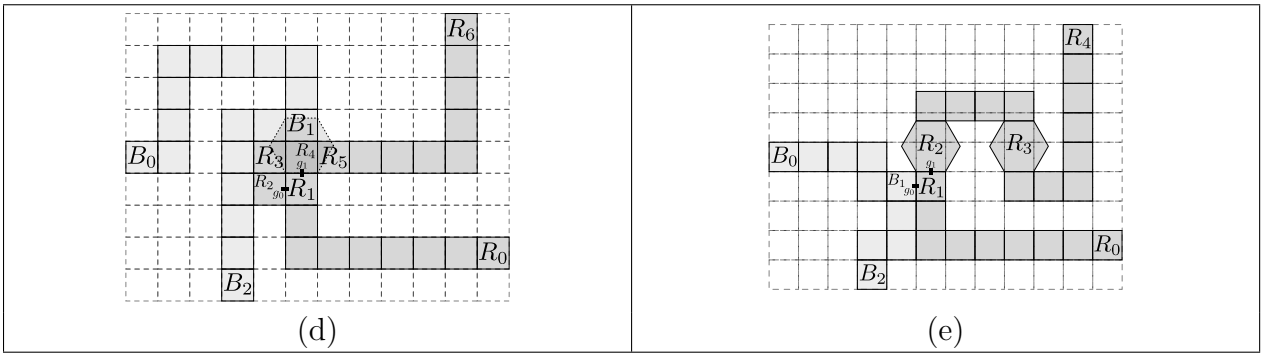


Table 4.4: Configurations for normalized on-grid bit-reading gadgets that can be used for 2-shaped systems using whose tiles have the shape of a square and a regular hexagon. (a) represents a 0, and (b) represents a 1.

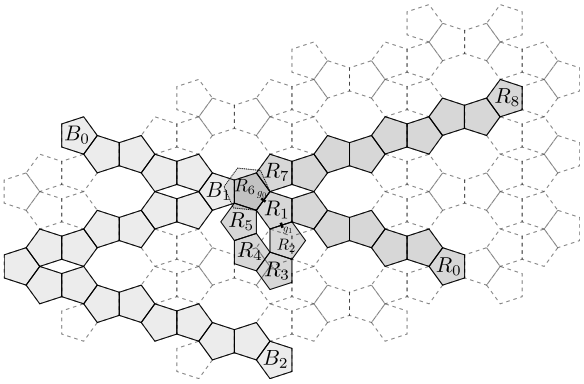


Figure 4.55: Bit-reading gadget configuration for tiles with the shape of either a pentagon or a hexagon. This figure depicts a configuration of polygonal tiles which represents a 0, while Figure 4.56 depicts a configuration of polygonal tiles which represents a 1.

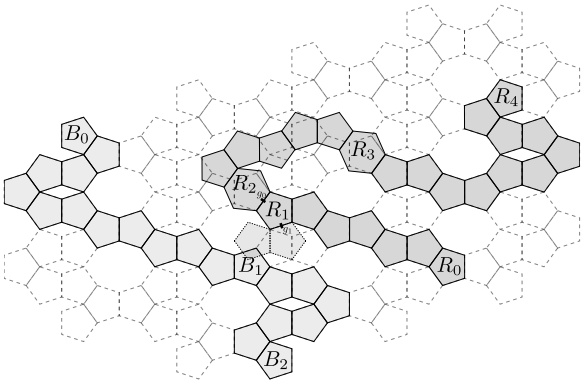


Figure 4.56: This figure depicts a configuration of polygonal tiles with the shape of either a pentagon or a hexagon which represents a 0.

4.10.3 Single shaped systems with equilateral polygonal tiles

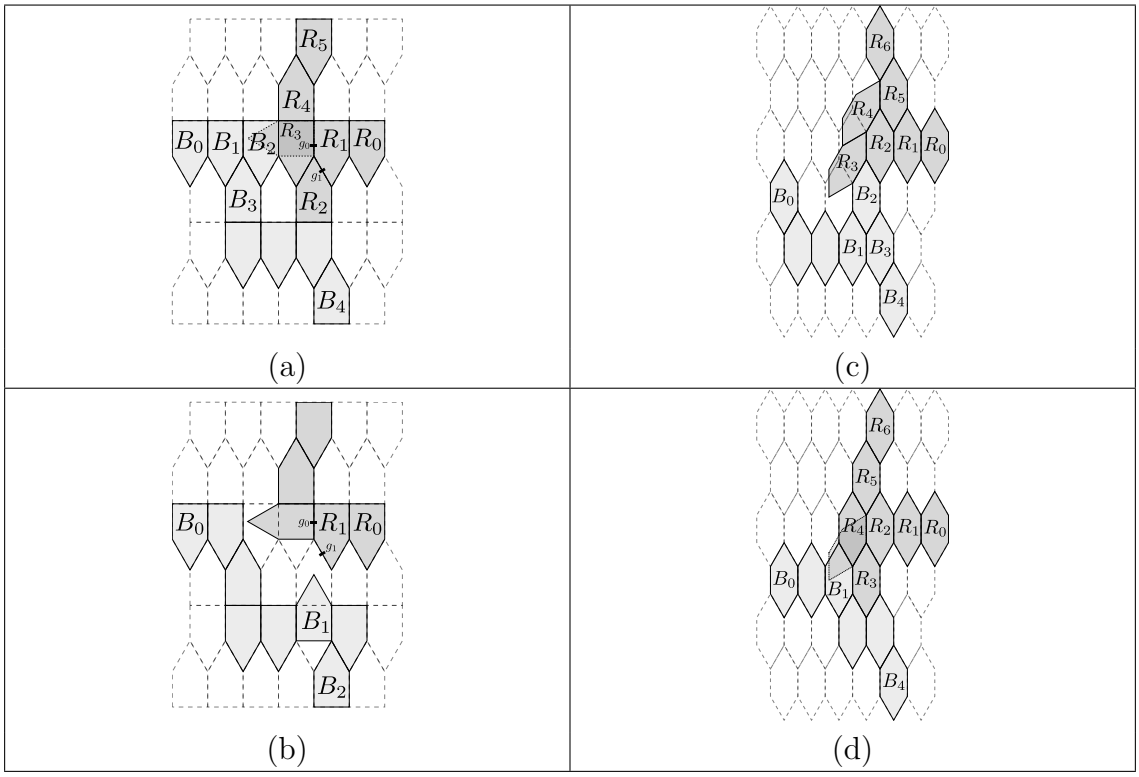


Table 4.5: Configurations of for normalized on-grid bit-reading gadgets that can be used for 1-shaped systems using whose tiles have the shape of a particular equilateral pentagon ((a) and (b)) or a particular equilateral hexagon ((c) and (d)).

Chapter 5

Conclusion and future work

We have introduced the polygonal TAM and proved two main results about simulation by directed tile assembly systems. The first result we proved answers an open question posed in [17] and shows that there is no universal simulator in the aTAM for the class of directed aTAM systems which is itself always directed. This reveals the essential role that nondeterminism plays in universal simulation in the aTAM. For the simulation of some directed systems, a universal simulator will face a situation where in order to faithfully simulate the target system, necessary race conditions require the simulator to be undirected. Our second set of results prove a positive result about the simulation of Turing machines by non-cooperative polygonal TAM systems and a negative result about the simulation of Turing machines by non-cooperative polygonal TAM systems using known techniques. In cooperative systems, the constraint that two glues on a tile must match the glues of neighboring tiles allows bit-reading which in turn provides a mechanism for simulating Turing machines. In non-cooperative systems, tiles bind to an assembly if one or more glues match. This means the mechanism used for bit-reading in cooperative systems cannot be used in non-cooperative systems. As an alternative, our positive result shows that geometry in conjunction with information stored in glues can be used to mimic bit-reading in systems where the shape of tiles are regular polygons with seven or more sides. On the other hand, we show that bit-reading is impossible in non-cooperative polygonal TAM systems where the shape of the tiles are regular polygons with six or less sides.

5.1 Intrinsic universality of systems with varying levels of nondeterminism

In this dissertation, we examined whether a universal simulator required undirectedness to simulate the class of directed systems. Even in directed systems, nondeterminism can still be present. A directed system can have several sequences of intermediate assemblies which lead to the final terminal assembly. Thus, a further restriction we can place on non-

determinism in tile assembly systems is that there is only one assembly sequence which leads to the terminal assembly. This is equivalent to saying that at any given time during the assembly process of a directed system, there is exactly one tile which can attach to the assembly. The following questions about the role nondeterminism plays in simulation remain open. (1) Are single assembly sequence systems intrinsically universal? (2) Is there a directed simulator for the class of single assembly sequence systems? (3) A zig-zag system is a sub-class of single assembly sequence systems in which growth must occur in a zig-zag like manner. It is known that this class of systems is computationally universal. There is a straight forward proof to see that zig-zag systems are not intrinsically universal when macrotiles are restricted to be square, but the proof fails to hold for rectangular macrotiles. Is the class of zig-zag systems intrinsically universal when allowing for rectangular macrotiles? (4) Our impossibility result fundamentally relies on the simulator assembling in the plane. Is the 3D directed aTAM intrinsically universal?

By answering these questions, we could gain further insights into the role nondeterminism plays in self-assembly.

5.2 Simulation in the polygonal TAM

We now present two separate directions of further research in the polygonal TAM. The first direction builds on the results presented in this dissertation and is concerned with further exploring the requirements for a polygonal TAM system to simulate a Turing machine. The second direction involves simulating polygonal TAM systems with other polygonal TAM systems to understand the role tile shape plays in self-assembly. The following questions remain open. (1) What are the necessary and sufficient conditions required for a polygon P so that there exists a non-cooperative system containing tiles of shape P which has a bit reading-gadget? Recall that this would imply the class of systems containing tiles of shape P is capable of universal computation. (2) How should simulation be defined in the polygonal TAM? The definition of simulation should be robust enough to allow for sys-

tems with different shape tiles to potentially simulate each other. Recall that the aTAM definition of simulation allows for fuzz (which is necessary to allow the simulator to imitate he local interactions which occur in the target system). A key part of this definition will involve defining fuzz in polygonal TAM systems. (3) What does the simulation landscape look like in the polygonal TAM? Let P be a regular polygon with n sides and let P' be a regular polygon with less than n sides. More specifically, can the class of polygonal TAM systems with tiles of shape P' be simulated by the class of systems with tiles of shape P? What about vice versa? Is there a hierarchy of simulation based on the shapes of the tiles in the systems or are the powers of these classes of systems disjoint?

References

[1] Ebbe S. Andersen, Mingdong Dong, Morten M. Nielsen, Kasper Jahn, Ramesh Subramani, Wael Mamdouh, Monika M. Golas, Bjoern Sander, Holger Stark, Cristiano L. P. Oliveira, Jan S. Pedersen, Victoria Birkedal, Flemming Besenbacher, Kurt V. Gothelf, and Jorgen Kjems. Self-assembly of a nanoscale dna box with a controllable lid. *Nature*, 459(7243):73–76, May 2009.

[2] Nathaniel Bryans, Ehsan Chiniforooshan, David Doty, Lila Kari, and Shinnosuke Seki. The power of nondeterminism in self-assembly. In *SODA 2011: Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 590–602. SIAM, 2011.

[3] Nathaniel Bryans, Ehsan Chiniforooshan, David Doty, Lila Kari, and Shinnosuke Seki. The power of nondeterminism in self-assembly. *Theory of Computing*, 9:1–29, 2013.

[4] Matthew Cook, Yunhui Fu, and Robert T. Schweller. Temperature 1 self-assembly: Deterministic assembly in 3D and probabilistic assembly in 2D. In *SODA 2011: Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2011.

[5] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[6] E. D. Demaine, M. L. Demaine, S. P. Fekete, M. J. Patitz, R. T. Schweller, A. Winslow, and D. Woods. One tile to rule them all: Simulating any tile assembly system with a single universal tile. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP 2014),* IT University of Copenhagen, Denmark, July 8-11, 2014, volume 8572 of *LNCS*, pages 368–379, 2014.

[7] Erik D. Demaine, Matthew J. Patitz, Trent A. Rogers, Robert T. Schweller, Scott M. Summers, and Damien Woods. The two-handed assembly model is not intrinsically universal. In *40th International Colloquium on Automata, Languages and Programming, ICALP 2013, Riga, Latvia, July 8-12, 2013*, Lecture Notes in Computer Science. Springer, 2013.

[8] David Doty. Randomized self-assembly for exact shapes. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 85–94. IEEE, 2009.

[9] David Doty, Jack H. Lutz, Matthew J. Patitz, Robert T. Schweller, Scott M. Summers, and Damien Woods. The tile assembly model is intrinsically universal. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science*, FOCS 2012, pages 302–310, 2012.

[10] David Doty, Matthew J. Patitz, and Scott M. Summers. Limitations of self-assembly at temperature 1. *Theoretical Computer Science*, 412:145–158, 2011.

[11] B. Durand and Zs. Róka. The game of life: universality revisited. In M. Delorme and J. Mazoyer, editors, *Cellular Automata*. Kluwer, 1999.

[12] Constantine Glen Evans. *Crystals that count! Physical principles and experimental investigations of DNA tile self-assembly.* PhD thesis, California Institute of Technology, 2014.

[13] Sándor P. Fekete, Jacob Hendricks, Matthew J. Patitz, Trent A. Rogers, and Robert T. Schweller. Universal computation with arbitrary polyomino tiles in non-cooperative self-assembly. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015), San Diego, CA, USA* January 4-6, 2015, pages 148–167, 2015.

[14] Tyler Fochtman, Jacob Hendricks, Jennifer E. Padilla, Matthew J. Patitz, and Trent A. Rogers. Signal transmission across tile assemblies: 3d static tiles simulate active self-assembly by 2d signal-passing tiles. *Natural Computing*, 14(2):251–264, 2015.

[15] Bin Fu, Matthew J. Patitz, Robert T. Schweller, and Robert Sheline. Self-assembly with geometric tiles. In *Proceedings of the 39th International Colloquium on Automata, Languages and Programming*, ICALP, pages 714–725, 2012.

[16] Oscar Gilbert, Jacob Hendricks, Matthew J. Patitz, and Trent A. Rogers. Computing in continuous space with self-assembling polygonal tiles. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016), Arlington, VA, USA* January 10-12, 2016, pages 937–956, 2016.

[17] Jacob Hendricks. *Simulation in Algorithmic Self-assembly.* PhD thesis, University of Arkansas, 2015.

[18] Jacob Hendricks, Matthew J. Patitz, and Trent A. Rogers. The simulation powers and limitations of higher temperature hierarchical self-assembly systems. In *7th International Conference on Machines, Computations and Universality (MCU'15), (9-11 September, 2015, Eastern Mediterranean University, Famagusta, North Cyprus)*, pages 149–163.

[19] Jacob Hendricks, Matthew J. Patitz, and Trent A. Rogers. Doubles and negatives are positive (in self-assembly). In *Proceeding of Unconventional Computation and Natural Computation 2014 (UCNC 2014),* University of Western Ontario, London, Ontario, Canada, 7/14/2014 - 7/18/2014, pages 190–202, 2014.

[20] Jacob Hendricks, Matthew J. Patitz, Trent A. Rogers, and Scott M. Summers. The power of duples (in self-assembly): It's not so hip to be square. In *Computing and Combinatorics - 20th International Conference, (COCOON) 2014, Atlanta, GA, USA, August 4-6, 2014. Proceedings*, pages 215–226, 2014.

[21] Jacob Hendricks, Matthew J. Patitz, Trent A. Rogers, and Scott M. Summers. The power of duples (in self-assembly): It's not so hip to be square. *Theoretical Computer Science*, 2015.

[22] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, October 1988.

[23] Ming-Yang Kao and Robert T. Schweller. Randomized self-assembly for approximate shapes. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (1)*, volume 5125 of *Lecture Notes in Computer Science*, pages 370–384. Springer, 2008.

[24] Jin-Woo Kim, Jeong-Hwan Kim, and Russell Deaton. Dna-linked nanoparticle building blocks for programmable matter. *Angewandte Chemie International Edition*, 50(39):9185–9190, 2011.

[25] Grégory Lafitte and Michael Weiss. Simulations between tilings. In *Conference on Computability in Europe (CiE 2008), local proceedings*, pages 264–273, 2008.

[26] James I. Lathrop, Jack H. Lutz, Matthew J. Patitz, and Scott M. Summers. Computability and complexity in self-assembly. *Theory Comput. Syst.*, 48(3):617–647, 2011.

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

[28] Ján Maňuch, Ladislav Stacho, and Christine Stoll. Two lower bounds for self-assemblies at temperature 1. *Journal of Computational Biology*, 17(6):841–852, 2010.

[29] Pierre-Étienne Meunier, Matthew J. Patitz, Scott M. Summers, Guillaume Theyssier, Andrew Winslow, and Damien Woods. Intrinsic universality in tile self-assembly requires cooperation. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA 2014), (Portland, OR, USA, January 5-7, 2014)*, pages 752–771, 2014.

[30] Nicolas Ollinger. Intrinsically universal cellular automata. In *The Complexity of Simple Programs, in Electronic Proceedings in Theoretical Computer Science*, volume 1, pages 199–204, 2008.

[31] Matthew J. Patitz, Robert T. Schweller, and Scott M. Summers. Exact shapes and turing universality at temperature 1 with a single negative glue. In *Proceedings of the 17th international conference on DNA computing and molecular programming*, DNA'11, pages 175–189, 2011.

[32] Matthew J. Patitz and Scott M. Summers. Self-assembly of decidable sets. *Natural Computing*, 10(2):853–877, 2011.

[33] José D. P. Rolim and Sheila A. Greibach. A note on the best-case complexity. *Inf. Process. Lett.*, 30(3):133–138, 1989.

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

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

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

[37] Paul WK Rothemund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of dna sierpinski triangles. *PLoS biology*, 2(12):e424, 2004.

[38] Rebecca Schulman, Bernard Yurke, and Erik Winfree. Robust self-replication of combinatorial information via crystal growth and scission. *Proc Natl Acad Sci U S A*, 109(17):6405–10, 2012.

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

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

[41] Damien Woods. Intrinsic universality and the computational power of self-assembly. In *MCU: Proceedings of Machines, Computations and Universality*, volume 128, pages 16–22, Univ. of Zürich, Switzerland. Sept. 9-12, 2013. Open Publishing Association.