7-2022

Encoding Color Sequences in Active Tile Self-Assembly

Sonya Cirlos
The University of Texas Rio Grande Valley

ENCODING COLOR SEQUENCES IN ACTIVE SELF-ASSEMBLY

A Thesis

by

SONYA CIRLOS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: Computer Science

The University of Texas Rio Grande Valley

July 2022

ENCODING COLOR SEQUENCES IN ACTIVE SELF-ASSEMBLY

A Thesis
by
SONYA CIRLOS

COMMITTEE MEMBERS

Dr. Robert Schweller
Chair of Committee

Dr. Tim Wylie
Committee Member

Dr. Bin Fu
Committee Member

July 2022

ABSTRACT

Cirlos, Sonya, Encoding Color Sequences in Active Self-Assembly. Master of Science (MS), July, 2022, 47 pp., 1 table, 15 figures, 45 references.

Constructing patterns is a well-studied problem in both theoretical and experimental self-assembly with much of the work focused on multi-staged assembly. In this paper, we study building 1D patterns in a model of active self assembly: Tile Automata. This is a generalization of the 2-handed assembly model that borrows the concept of state changes from Cellular Automata. In this work we further develop the model by partitioning states as colors and show lower and upper bounds for building patterned assemblies based on an input pattern. Our first two sections utilize recent results to build binary strings along with Turing machine constructions to get Kolmogorov optimal state complexity for building patterns in Tile Automata, and show nearly optimal bounds for one case. For affinity strengthening Tile Automata, where transitions can only increase affinity so there is no detachment, we focus on scaled patterns based on Space Bounded Kolmogorov Complexity. Finally, we examine the affinity strengthening freezing case providing an upper bound based on the minimum context-free grammar. This system utilizes only one dimensional assemblies and has tiles that do not change color.

DEDICATION

To my parents Michael and Irma, and my siblings Jasmine, Marcus, Micah, Memo, and Blue. I am forever indebted for their unconditional love and support.

ACKNOWLEDGMENTS

I am grateful for the guidance, patience, and support from my mentors Dr. Robert Schweller, Dr. Tim Wylie, and Dr. Bin Fu these past two years. I would like to thank to my research team Tim, David, Elise, Michael, and Andrew for providing me with help and catching me up on the world on computer science theory. To my best friends Stephany and Catalina; I would not have pursued a Masters in Computer Science without their push. I would also like to thank Valentin for his continued motivation and support as I strived to finish the last year of my degree.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER I

INTRODUCTION

Designing and studying different self-assembly systems has quickly grown in biological research because of their simplicity to create DNA level systems to help understand and advance experimental techniques. Self-Assembly is the natural process by which small particles randomly agitate and combine through rules and local interactions to assemble into complex structures. Today, there are many models of tile self-assembly, each focusing on different aspects of self-assembling systems. Some models of self-assembly include the Abstract Tile Assembly Model (aTAM) [41], the Staged self-assembly model [16], and the 2-handed Assembly Model [10]. In this thesis, I focus on the Tile Automata model [12] and shape and pattern building results in different versions Tile Automata models.

1.1 Overview

Chapter II of this thesis, includes our research group's recent published manuscript: Building Squares with Optimal State Complexity in Restricted Active Self-Assembly. Here, we use a seeded version of the Tile Automata model to prove optimal complexity bounds for string, squares and rectangle constructions. This paper was accepted to a conference where it won the award of Best Student Paper and was chosen to be published in the Journal of Computer and Systems Science. Chapter III introduces restricted version of Tile Automata system that can simulate a Context- Free Grammar to build a 1D patterned assembly that represents a string from a given grammar.

1.2 Squares

Since the start of the self-assembly field, researchers developed several systems that have yielded many interesting results. One well-studied abstract system is known as Tile-Self Assembly. From this system, researchers began designing other models such as the abstract Tile Assembly Model (aTAM). ATAM is a subset of asynchronous well-known powerful model called cellular automata [20]. Finding the most optimal way to build certain benchmark shapes such as squares and rectangles in self-assembly [1, 37]. Due to experimental work in [14, 26, 40] active self assembly is a fast and growing research area. Because cellular automata is a very powerful system, researchers developed a model that meets the middle ground between aTAM and cellular automata. In [12], researchers designed a new model of self assembly called Tile Automata, that has attributes of both aTAM and cellular automata with restrictions that can limit their power. In Chapter II, I provide one recently published manuscript from our research team, where we use a seeded version of the Tile Automata model to prove optimal complexity bounds for string, squares and rectangle constructions.

This past year, I helped design and develop our group's Tile Automata simulator, AutoTile. AutoTile simulates a seeded Tile Automata system. AutoTile was developed by our research group of students to aid other self-assembly researchers in their studies and to confirm the results for our seeded Tile Automata constructions for strings, squares, and rectangles. To have simulator users create and edit a Tile Automata system, I designed and developed AutoTile's Editor window, where users can edit a system they uploaded to AutoTile to test their own seeded Tile Automata construction algorithms.

1.3 Patterns

In [17], Demaine and his group uncovered the power of the staged assembly model by developing a connection between the smallest context-free grammars and the staged self-assembly systems for one-dimensional strings and assemblies. The 1D staged self-assembly model minimizes components and adds restrictions to its system to make any shape in constant number of particle

types, and has drawn interest from experimenters because of its natural and practical design. This staged self-assembly model can successfully simulate a Context-Free Grammar deriving a particular string to construct an assembly with a label of the string. In this work, we explore the strength of Tile Automata by studying the problem of finding the smallest tile automata system with minimum number of states, producing a one-dimensional patterned line assembly and comparing it to the well-known problem of finding the minimum context-free grammar. In Chapter III, I introduce a restricted version of Tile Automata, where states are given a designated color attribute and rule that a state must remain "color-locked" (i.e. states cannot change color). We prove that this restricted Tile Automata model is at least as strong as Context-Free Grammars in describing strings.

CHAPTER II

SHAPE BUILDING IN SEEDED MODELS

2.1 Introduction

Self-assembly is the process by which simple elements in a system organize themselves into more complex structures based on a set of rules that govern their interactions. These types of systems occur naturally and can be easily constructed artificially to offer many advantages when building micro or nanoscale objects. One abstraction of these systems that has yielded interesting results is Tile Self-Assembly.

In the abstract Tile Assembly Model (aTAM) [41], the elements of a system are represented using labeled unit squares called tiles. A system is initialized with a seed (a tile or assembly) that grows as other single tiles attach until there are no more valid attachments. The behavior of a system can then be programmed, using the interactions of tiles, and is known to be capable of Turing Computation [41], is Intrinsically Universal [19], and can assemble general scaled shapes [39]. However, many of these results utilize a concept called *cooperative binding*, where a tile must attach to an assembly using the interaction from two other tiles. Unlike with cooperative binding, the non-cooperative aTAM is not Intrinsically Universal [30, 32] and more recent work has shown that it is not capable of Turing Computation [31]. Many extensions of this model increase the power of non-cooperative systems [6, 21, 23, 27, 28, 36].

One recent model of self-assembly is Tile Automata [12]. This model marries the concept of state changes from Cellular Automata [24, 33, 45] and the assembly process from the 2-Handed Assembly model (2HAM) [10]. Previous work [5, 11, 12] has explored Tile Automata as a unifying

model for comparing the relative powers of the many different Tile Assembly models. The complexity of verifying the behavior of systems along with their computational power was studied in [9]. Many of these works impose additional experimentally motivated limitations on the Tile Automata model that help connect the model and its capabilities to potential molecular implementations, such as using DNA assemblies with sensors to assemble larger structures [26], building spacial localized circuits on DNA origami [14], or DNA walkers that sort cargo [40].

In this paper, we explore the aTAM generalized with state changes; we define our producible assemblies as what can be grown by attaching tiles one at a time to a seed tile or performing transition rules, which we refer to as seeded Tile Automata. This is a bounded version of Asynchronous Cellular Automata [20]. Reachability problems, which are similar to verification problems in self-assembly, have been studied with many completeness results [18]. Further, the freezing property used in this and previous work also exists in Cellular Automata [25, 34].[1] Freezing is defined differently in Cellular Automata by requiring that there exists an ordering to the states.

While Tile Automata has many possible metrics, we focus on the number of states needed to uniquely assemble $n \times n$ squares at the smallest constant temperature, $\tau = 1$. We achieve optimal bounds in three versions of the model with varying restrictions on the transition rules. Our results, along with previous results in the aTAM, are outlined in Table 2.1.

2.1.1 Previous Work

In the aTAM, the number of tile types needed, for nearly all n, to construct an $n \times n$ square is $\Theta(\frac{\log n}{\log n \log n})$ [1, 37] with temperature $\tau = 2$ (row 2 of Table 2.1). The same lower bounds hold for $\tau = 1$ (row 1 of Table 2.1). The run time of this system was also shown to be optimal $\Theta(n)$ [1]. Other bounds for building rectangles were shown in [2]. While no tighter bounds[2] have been shown for $n \times n$ squares at $\tau = 1$ in the aTAM, generalizations to the model that allow (just-barely) 3D growth have shown an upper bound of $\mathcal{O}(\log n)$ for tile types needed [15]. Recent work in [22]

[1] We would like to thank a reviewer for bringing these works to our attention.

[2] Other than trivial $\mathcal{O}(n)$ bounds.

Model	τ	$n \times n$ Squares		
		Lower	**Upper**	**Theorem**
aTAM	1	$\Omega(\frac{\log n}{\log \log n})$	$\mathcal{O}(n)$	[37], [1]
aTAM	2	$\Theta(\frac{\log n}{\log \log n})$		[37], [1]
Flexible Glue aTAM	2	$\Theta(\log^{\frac{1}{2}} n)$		[2]
Seeded TA Det.	1	$\Theta((\frac{\log n}{\log \log n})^{\frac{1}{2}})$		Thm. 2.2.2, 2.5.1
Seeded TA ST	1	$\Theta(\log^{\frac{1}{3}} n)$		Thm. 2.2.4, 2.5.1
Seeded TA	1	$\Theta(\log^{\frac{1}{4}} n)$		Thm. 2.2.3, 2.5.1

Table 2.1: Bounds on the number of states for $n \times n$ squares in the Abstract Tile Assembly model, with and without cooperative binding, and the seeded Tile Automata model with our transition rules. ST stands for Single-Transition.

shows improved upper and lower bounds on building thin rectangles in the case of $\tau = 1$ and in (just-barely) 3D.

Other models of self-assembly have also been shown to have a smaller tile complexity, such as the staged assembly model [13, 16] and temperature programming [**?**]. Investigation into different active self-assembly models have also explored the run time of systems [38, 43].

2.1.2 Our Contributions

In this work, we explore building an important benchmark shape, squares, in non-cooperative seeded Tile Automata. We also consider only affinity-strengthening transition rules that remove the ability for an assembly to break apart. Our results are shown in Table 2.1.

We start in Section 2.2 by proving lower bounds for building $n \times n$ squares based on three different transition rule restrictions. The first is nondeterministic or general seeded Tile Automata, where there are no restrictions and a pair of states may have multiple transition rules. The second is Single-Transition rules where only one tile may change states in a transition rule, but we still allow multiple rules for each pair of states. The last restriction, Deterministic, is the most restrictive where each pair of states may only have one transition rule (for each direction).

In Section 2.3, we use Transition Rules to optimally encode strings in the various versions of the model. We use these encodings as gadgets to seed the future constructions. We show how

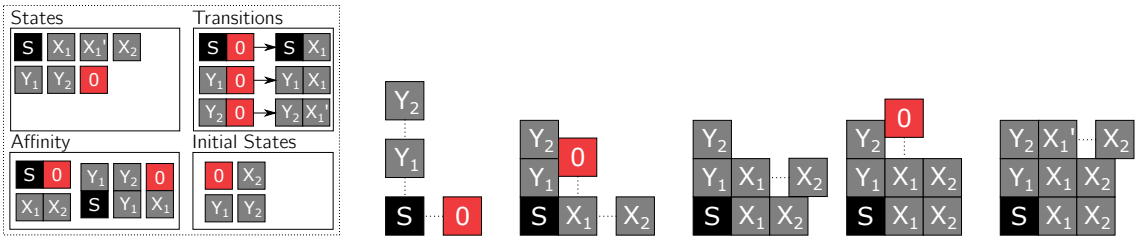


Figure 2.1: (a) Example of a Tile Automata system, it should be noted that $\tau = 1$ and state S is our seed. (b) A walkthrough of our example Tile Automata system building the 3×3 square it uniquely produces. We use dotted lines throughout our paper to represent tiles attaching to one another.

to build optimal state complexity rectangles in Section 2.4, and finally optimal state complexity squares in Section 2.5. Future work is discussed in Section 2.6.

AutoTile. To test our constructions, we developed AutoTile, a seeded Tile Automata simulator. Each system discussed in the paper is currently available for simulation. AutoTile is available at `https://github.com/asarg/AutoTile`.

2.1.3 The Tile Automata Model

Here, we define and investigate the *Seeded Tile Automata* model, which differs by only allowing single tile attachments to a growing seed similar to the aTAM.

Seeded Tile Automata. A Seeded Tile Automata system is a 6-tuple $\Gamma = \{\Sigma, \Lambda, \Pi, \Delta, s, \tau\}$ where Σ is a set of states, $\Lambda \subseteq \Sigma$ a set of *initial states*, Π is an *affinity function*, Δ is a set of *transition rules*, s is a stable assembly called the *seed* assembly, and τ is the *temperature* (or threshold). Our results use the most restrictive version of this model where s is a single tile.

Attachment Step. A tile $t = (\sigma, p)$ may attach to an assembly A at temperature τ to build an assembly $A' = A \bigcup t$ if A' is τ-stable and $\sigma \in \Lambda$. We denote this as $A \rightarrow_{\Lambda,\tau} A'$.

Transition Step. An assembly A is transitionable to an assembly A' if there exists two neighboring tiles $t_1 = (\sigma_1, p_1), t_2 = (\sigma_2, p_2) \in A$ (where t_1 is the west or north tile) such that there exists a transition rule in Δ with the first pair being (σ_1, σ_2) and $A' = (A \setminus \{t_1, t_2\}) \bigcup \{t_3 = (\sigma_3, p_1), t_4 = (\sigma_4, p_2)\}$. We denote this as $A \rightarrow_{\Delta} A'$.

Producibles. We refer to both attachment steps and transition steps as production steps, we define $A \to_* A'$ as the transitive closure of $A \to_{\Lambda,\tau} A'$ and $A \to_\Delta A'$. The set of *producible assemblies* for a Tile Automata system $\Gamma = \{\Sigma, \Lambda, \Pi, \Delta, s, \tau\}$ is written as $PROD(\Gamma)$. We define $PROD(\Gamma)$ recursively as follows,

- $s \in PROD(\Gamma)$
- $A' \in PROD(\Gamma)$ if $\exists A \in PROD(\Gamma)$ such that $A \to_{\Lambda,\tau} A'$.
- $A' \in PROD(\Gamma)$ if $\exists A \in PROD(\Gamma)$ such that $A \to_\Delta A'$.

Terminal Assemblies. The set of terminal assemblies for a Tile Automata system $\Gamma = \{\Sigma, \Lambda, \Pi, \Delta, \tau\}$ is written as $TERM(\Gamma)$. This is the set of assemblies that cannot grow or transition any further. Formally, an assembly $A \in TERM(\Gamma)$ if $A \in PROD(\Gamma)$ and there does not exists any assembly $A' \in PROD(\Gamma)$ such that $A \to_{\Lambda,\tau} A'$ or $A \to_\Delta A'$. A Tile Automata system $\Gamma = \{\Sigma, \Lambda, \Pi, \Delta, s, \tau\}$ *uniquely* assembles an assembly A if $A \in TERM(\Gamma)$, and for all $A' \in PROD(\Gamma), A' \to_* A$.

2.1.4 Limited Model Reference

We explore an extremely limited version of seeded TA that is affinity-strengthening, freezing, and may be a single-transition system. We investigate both deterministic and non-deterministic versions of this model.

Affinity Strengthening. We only consider transitions rules that are affinity strengthening, meaning for each transition rule $((\sigma_1, \sigma_2), (\sigma_3, \sigma_4), d)$, the bond between (σ_3, σ_4) must be at least the strength of (σ_1, σ_2). Formally, $\Pi(\sigma_3, \sigma_4, d) \geq \Pi(\sigma_1, \sigma_2, d)$. This ensures that transitions may not induce cuts in the bond graph.

In the case of non-cooperative systems ($\tau = 1$), the affinity strength between states is always 1 so we may refer to the affinity function as an affinity set Λ_s, where each affinity is a 3-pule (σ_1, σ_2, d).

Freezing. Freezing systems were introduced with Tile Automata. A freezing system simply means that a tile may transition to any state only once. Thus, if a tile is in state A and transitions to another state, it is not allowed to ever transition back to A.

Deterministic vs. Nondeterministic. For clarification, a deterministic system in TA has only one possible production step at a time, whether that be an attachment or a state transition. A nondeterministic system may have many possible production steps and any choice may be taken.

Single-Transition System. We restrict our TA system to only use single-transition rules. This means that for each transition rule one of the states may change, but not both. It should be noted that we still allow Nondeterminism in this system.

2.2 State Space Lower Bounds

Let $p(n)$ be a function from the positive integers to the set $\{0,1\}$, informally termed a *proposition*, where 0 denotes the proposition being false and 1 denotes the proposition being true. We say a proposition $p(n)$ holds for *almost all* n if $\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} p(i) = 1$.

Lemma 2.2.1. Let U be a set of TA systems, b be a one-to-one function mapping each element of U to a string of bits, and ε a real number from $0 < \varepsilon < 1$. Then for almost all integers n, any TA system $\Gamma \in U$ that uniquely assembles either an $n \times n$ square or a $1 \times n$ line has a bit-string of length $|b(\Gamma)| \geq (1-\varepsilon)\log n$.

Proof. For a given $i \geq 1$, let $M_i \in U$ denote the TA system in U with the minimum value $|b(M_i)|$ over all systems in U that uniquely assembly an $i \times i$ square or $1 \times i$ line, and let M_i be undefined if no such system in U builds such a shape. Let $p(i)$ be the proposition that $|b(M_i)| \geq (1-\varepsilon)\log i$. We show that $\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} p(i) = 1$. Let $R_n = \{M_i | 1 \leq i \leq n, |b(M_i)| < (1-\varepsilon)\log n\}$. Note that $n - |R_n| \leq \sum_{i=1}^{n} p(i)$. By the pigeon-hole principle, $|R_n| \leq 2^{(1-\varepsilon)\log n} = n^{(1-\varepsilon)}$. Therefore,

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} p(i) \geq \lim_{n\to\infty} \frac{1}{n}(n - |R_n|) \geq \lim_{n\to\infty} \frac{1}{n}(n - n^{1-\varepsilon}) = 1.$$

□

Theorem 2.2.2 (Deterministic TA)**.** For almost all n, any Deterministic Tile Automata system that uniquely assembles either a $1 \times n$ line or an $n \times n$ square contains $\Omega(\frac{\log n}{\log \log n})^{\frac{1}{2}}$ states.

Proof. We can create a one-to-one mapping $b(\Gamma)$ from any deterministic TA system to bit-strings in the following way. Let S denote the set of states in a given system. We encode the state set in $\mathcal{O}(\log |S|)$ bits, we encode the affinity function in a $|S| \times |S|$ table of strengths in $\mathcal{O}(|S|^2)$ bits (assuming a constant bound on bonding thresholds), and we encode the rules of the system in an $|S| \times |S|$ table mapping pairs of rules to their unique new pair of rules using $\mathcal{O}(|S|^2 \log |S|)$ bits, for a total of $\mathcal{O}(|S|^2 \log |S|)$ bits to encode any $|S|$ state system.

Let Γ_n denote the smallest state system that uniquely assembles an $n \times n$ square (or similarly a $1 \times n$ line), and let S_n denote the state set. By Lemma 2.2.1, $|b(\Gamma_n)| \geq (1-\varepsilon) \log n$ for almost all n, and so $|S_n|^2 \log |S_n| = \Omega(\log n)$ for almost all n. We know that $|S_n| = \mathcal{O}(\log n)$, so for some constant c, $|S_n| \geq c(\frac{\log n}{\log \log n})^{\frac{1}{2}}$ for almost all n. □

Theorem 2.2.3 (Nondeterministic TA)**.** For almost all n, any Tile Automata system (in particular any Nondeterministic system) that uniquely assembles either a $1 \times n$ line or an $n \times n$ square contains $\Omega(\log^{\frac{1}{4}} n)$ states.

Theorem 2.2.4 (Single-Transition TA)**.** For almost all n, any Single-Transition Tile Automata system that uniquely assembles either a $1 \times n$ line or an $n \times n$ square contains $\Omega(\log^{\frac{1}{3}} n)$ states.

2.3 String Unpacking

A key tool in our constructions is the ability to build strings efficiently. We do so by encoding the string in the transition rules.

Definition 2.3.1 (String Representation)**.** An assembly A over states Σ represent a string S over a set of symbols U if there exists a mapping from the elements of U to the elements of Σ and a $1 \times |S|$ (or $|S| \times 1$) subassembly $A' \sqsubset A$, such that the state of the i^{th} tile of A' maps to the i^{th} symbol of S for all $0 \leq i \leq |S|$.

2.3.1 Deterministic Transitions

We start by showing how to encode a binary string of length n in a set of (freezing) transition rules that take place on a $2 \times (n+2)$ rectangle that will print the string on its right side. We extend this construction to work for an arbitrary base string.

Overview Consider a system that builds a length n string. First, we create a rectangle of index states that is two wide as seen on the left side of Figure 2.5c. Each row has a unique pair of index states so each bit of the string is uniquely indexed. We divide the index states into two groups based on which column they are in, and which "digit" they represent. Let $r = \lceil n^{\frac{1}{2}} \rceil$. Starting with index states A_0 and B_0, we build a counter pattern with base r. We use $\mathcal{O}(n^{\frac{1}{2}})$ states shown in Figure 2.2 to build this pattern. We encode each bit of the string in a transition rule between the two states that index that bit. A table with these transition rules can be seen in Figure 2.5b.

The pattern is built in r sections of size $2 \times r$ with the first section growing off of the seed. The tile in state S_A is the seed. There is also a state S_B that has affinity for the right side of S_A. The building process is defined in the following steps for each section.

1. The states $S_B, 0_B, 1_B, \ldots, (r-1_B)$ grow off of S_B, forming the right column of the section. The last B state allows for a' to attach on its west side. a tiles attach below a' and below itself. This places a states in a row south toward the state S_A, depicted in Figure 2.3b.

2. Once a section is built, the states begin to follow their transition rules shown in Figure 2.4a. The a state transitions with seed state S_A to begin indexing the A column by changing state a to state 0_A. For $1 \leq y \leq n-2$, state a vertically transitions with the other y'_A states, incrementing the index by changing from state a to state $(y+1)_A$.

3. This new index state z_A propagates up by transitioning the a tiles to the state z_A as well. Once the z_A state reaches a' at the top of the column, it transitions a' to the state z'_A. Figure 2.4b presents this process of indexing the A column.

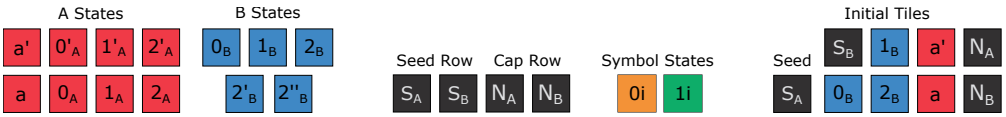


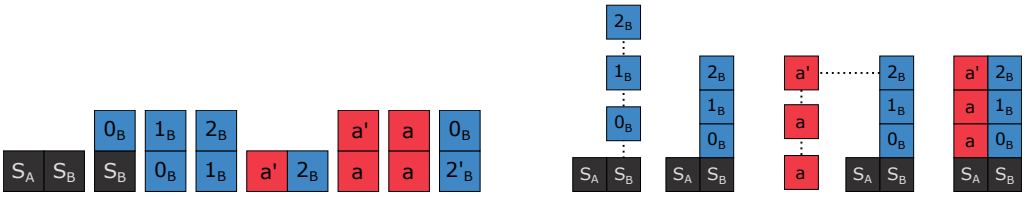
Figure 2.2: States to build a length-9 string in deterministic Tile Automata.

4. If $z < n-1$, there is a horizontal transition rule from states $(z'_A, n-1_B)$ to states $(z'_A, n-1'_B)$. The state 0_B attaches to the north of $n-1_B$ and starts the next section. If $z = n$, there does not exist a transition.

5. This creates an assembly with a unique state pair in each row as seen in the first column of Figure 2.5c.

States An example system with the states required to print a length-9 string are shown in Figure 2.2. The first states build the seed row of the assembly. The seed tile has the state S_A with initial tiles in state S_B. The index states are divided into two groups. The first set of index states, which we call the A index states, are used to build the left column. For each i, $0 \leq i < r$, we have the states i_A and i'_A. There are two states a and a', which exist as initial tiles and act as "blank" states that can transition to the other A states. The second set of index states are the B states. Again, we have r B states numbered from 0 to $r-1$, however, we do not have a prime for each state. Instead, there are two states $r-1'_B$ and $r-1''_B$, that are used to control the growth of the next column and the printing of the strings. The last states are the symbol states 0_S and 1_S, the states that represent the string.

Affinity Rules/Placing Section Here, we describe the affinity rules for building the first section. We later describe how this is generalized to the other $r-1$ sections. We walk through this process in Figure 2.3b. To begin, the B states attach in sequence above the tile S_B in the seed row. Assuming $r^2 = n$, n is a perfect square, the first state to attach is 0_B. 1_B attaches above this tile and so on. The last B state $r-1_B$ does not have affinity with 0_B, so the column stops growing. However, the state a' has affinity on the left of $r-1_B$ and can attach. a has affinity for the south side of a', so it attaches below. The a state also has a vertical affinity with itself. This grows the A column



(a) Affinity Rules for Initial Tiles (b) Process of Building a section

Figure 2.3: (a) Affinity rules to build each section. We only show affinity rules that are actually used in our system for initial tiles to attach, while our system would have more rules in order to meet the affinity strengthening restriction. (b) The B column attaches above the state S_B as shown by the dotted lines. The a' attaches to the left of 2_B and the other a states may attach below it until they reach S_A.

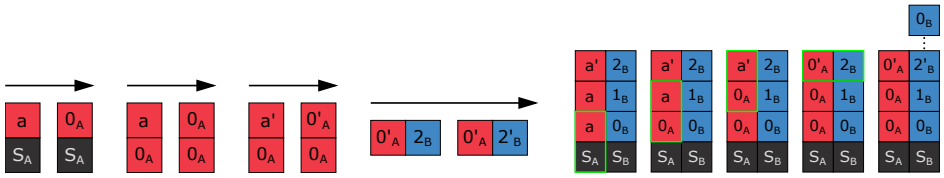
southward toward the seed row.

If n is not a perfect square, we start the index state pattern at a different value. We do so by finding the value $q = r^2 - n$. In general, the state i_B attaches above S_B for $i = q\%r$.

Transition Rules/Indexing A column Once the A column is complete and the last A state is placed above the seed, it transitions with S_A to 0_A (assuming $r^2 = n$). A has a vertical transition rule with i_A ($0 \leq i < r$) changing the state A to state i_A. This can be seen in Figure 2.4a, where the 0_A state is propagated upward to the A' state. The A' state also transitions when 0_A is below it, going from state A' to state $0'_A$. If n is not a perfect square, then A transitions to i_A for $i = \lfloor q/r \rfloor$.

Once the transition rules have finished indexing the A column if $i < r-1$, the last state i'_A transitions with $r-1_B$ changing the state $r-1_B$ to $r-1'_B$. This transition can be seen in Figure 2.4b. The new state $r-1'_B$ has an affinity rule allowing 0_B to attach above it allowing the next section to be built. When the state A is above a state j'_A, $0 \leq j < r-1$, it transitions with that state changing from state A to $j+1_A$, which increments the A index.

Look up After creating a $2 \times (n+2)$ rectangle, we can encode a length n string S into the transitions rules. Note that each row of our assembly consists of a unique pair of index states, which we call a *bit gadget*. Each bit gadget will *look up* a specific bit of our string and transition the B tile to a state representing the value of that bit.



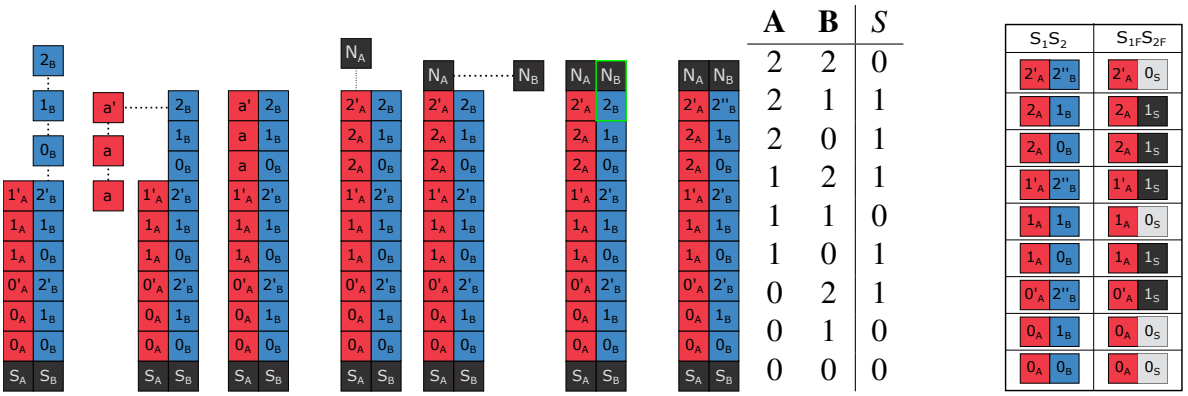
(a) Transition Rules to Index the first section (b) Process of Indexing A column

Figure 2.4: (a) The first transition rule used is takes place between the seed S_A and the a state changing to 0_A. The state 0_A changes the states north of it to 0_A or $0'_A$. Finally, the state $0'_A$ transitions with 2_B (b) Once the a states reach the seed row they transition with the state S_A to go to 0_A. This state propagates upward to the top of the section.

Figure 2.5b shows how to encode a string S in a table with two columns using r digits to index each bit. From this encoding, we create our transition rules. Consider the k^{th} bit of S (where the 0^{th} bit is the least significant bit) for $k = ir + j$. Add transition rules between the states i_A and j_B, changing the state j_B to either 0_S or 1_S based on the k^{th} bit of S. This transition rule is slightly different for the northmost row of each section as the state in the A column is i'_A. Also, we do not want the state in the B column, $r-1_B$, to prematurely transition to a symbol state. Thus, we have the two states $r-1'_B$ and $r-1''_B$. As mentioned, once the A column finishes indexing, it changes the state $r-1_B$ to state $r-1'_B$, allowing for 0_B to attach above it, which starts the next column. Once the state 0_B (or a symbol state) is above $r-1'_B$, there are no longer any possible undesired attachments, so the state transitions to $r-1''_B$, which has the transition to the symbol state.

The last section has a slightly different process as $r-1_B$ state will never have a 0_B attach above it, so we have a different transition rule. This alternate process is shown in Figure 2.5a. The state $r-1'_A$ has a vertical affinity with the cap state N_A. This state allows N_B to attach on its right side. This state transitions with $r-1_B$ below it, changing it directly to $r-1''_B$, allowing the symbol state to print.

Theorem 2.3.2. For any binary string s with length $n > 0$, there exists a freezing tile automata system Γ_s with deterministic transition rules, that uniquely assembles an $2 \times (n+2)$ assembly A_S that represents S with $\mathcal{O}(n^{\frac{1}{2}})$ states.



A	B	S
2	2	0
2	1	1
2	0	1
1	2	1
1	1	0
1	0	1
0	2	1
0	1	0
0	0	0

(a) Attaching Cap Row (b) Encoding of S (c) Transition Rules

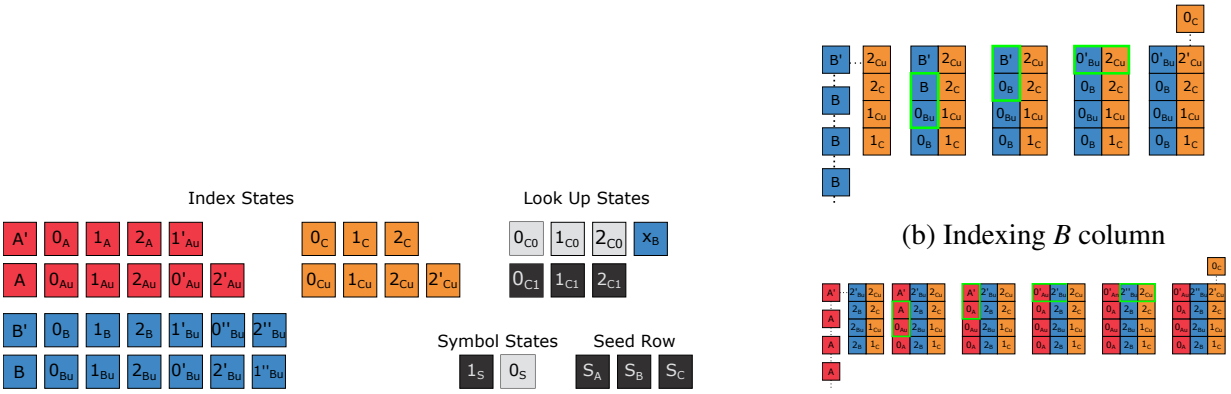
Figure 2.5: (a) Once the last section finishes building the state N_A attaches above $2'_A$. N_B then attaches to the assembly and transitions with 2_B changing it directly to $2''_B$ so the string may begin printing. (b) A table indexing the string $S = 011101100$ using two columns and base $|S|^{\frac{1}{2}}$. (c) Transition Rules to print S. We build an assembly where each row has a unique pair of index states in ascending order.

Arbitrary Base In order to optimally build rectangles, we first print arbitrary base strings. Here, we show how to generalize Theorem 2.3.2 to print base-b strings.

Corollary 2.3.3. For any base-b string S with length $n > 0$, there exists a freezing tile automata system Γ with deterministic transition rules, that uniquely assembles an $(n+2) \times 2$ assembly which represents S with $\mathcal{O}(n^{\frac{1}{2}} + b)$ states.

2.3.2 Nondeterministic Single-Transition Systems

For the case of Single-Transition systems, we use the same method from above but instead building bit gadgets that are of size 3×2. Expanding to 3 columns allows for a third index digit to be used giving us an upper bound of $\mathcal{O}(n^{\frac{1}{3}})$. The second row will be used for error checking which we will describe later in the section. This system utilizes Nondeterministic transitions, (two states may have multiple rules with the same orientation) and is non-freezing (a tile may repeat states). This system also contains cycles in its production graph, this implies the system may run indefinitely. We conjecture this system has a polynomial run time. Here, let $r = \lceil n^{\frac{1}{3}} \rceil$.



(a) States space for when $|S| = 27$

(b) Indexing B column

(c) Indexing the A column

Figure 2.6: (a) States needed to construct a length 27 string where $r = 3$. (b) The index 0 propagates upward by transitioning the tiles in the column to 0_B and 0_{Bu} and transitions a' to $0'_{Bu}$. The state $0'_{Bu}$ transitions with the state 2_{Cu}, changing the state 2_{Cu} to $2'_{Cu}$, which has affinity with 0_C to build the next section. These rules also exist for the index 1. (c) When the index state 2_B reaches the top of the section, it transitions b' to $2'_{Bu}$. This state does not transition with the C column and instead has affinity with the state a', which builds the A column downward. The index propagates up the A column in the same way as the B column. When the index state 0_A reaches the top of the section, it transitions the state $2'_B$ to $2''_B$. This state transitions with 2_{Cu} changing it to $2'_{Cu}$ allowing the column to grow.

Index States and Look Up States We generalize the method from above to start from a C column. The B column now behaves as the second index of the pattern and is built using B' and B as the A column was in the previous system. Once the B reaches the seed row, it is indexed with its starting value. This construction also requires bit gadgets of height 2, so we will use index states i_A, i_B, i_C and north index states i_{Au}, i_{Bu}, i_{Cu} for $0 \leq i < r$. This allows us to separate the two functions of the bit gadget into each row. The north row has transition rules to control the building of each section. The bottom row has transition rules that encode the represented bit.

In addition to the index states, we use $2r$ look up states, 0_{Ci} and 1_{Ci} for $0 \leq i < r$. These states are used as intermediate states during the look up. The first number (0 or 1) represents the value of the retrieved bit, while the second number represents the C index of the bit. The A and B indices of the bit will be represented by the other states in the transition rule.

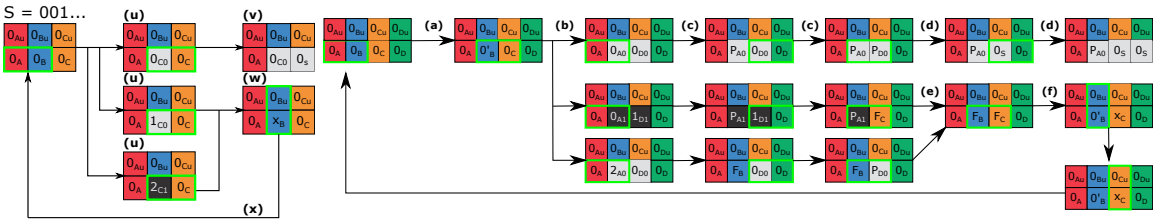
In the same way as the previous construction, we build the rightmost column first. We include the C index states as initial states and allow 0_C to attach above S_C. We include affinity rules to build the column northwards as follows starting with the southmost state $0_C, 0_{Cu}, 1_C, 1_{Cu}, \ldots, r-2_{Cu}, r-1_C, r-1_{Cu}$.

To build the other columns, the state b' can attach on the left of $r-1_{Cu}$. The state b is an initial state and attaches below b' and itself to grow downward toward the seed row. The state b transitions with the seed row as in the previous construction to start the column. However, we alternate between C states and Cu states. The state b above i_C transitions b to i_{Cu}. If b is above i_{Cu} it transitions to i_C. The state b' above state i_B transitions to i'_{Bu}. If $i < r-1$, the state i'_B and $r-1_{Cu}$ transition horizontally changing $r-1'_{Cu}$, which allows 0_C to attach above it to repeat the process. This is shown in Figure 2.6b.

The state a' attaches on the left of $r-1_{Cu}$. The A column is indexed just like the B column. For $0 \leq i < r-1$, the state i'_{Au} and $r-1'_{Bu}$ change the state $r-1'_{Bu}$ to $r-1''_{Bu}$. This state transitions with $r-1_{Cu}$, changing it to $r-1'_{Cu}$. See Figure 2.6c.

Bit Gadget Look Up The bottom row of each bit gadget has a unique sequence of states, again we use these index states to represent the bit indexed by the digits of the states. However, since we can only transition between two tiles at a time, we must read all three states in multiple steps. These steps are outlined in Figure 2.7a. The first transition takes place between the states i_A and j_B. We refer to these transition rules as look up rules. We have r look up rules between these states for $0 \leq k < r$ of these states that changes the state j_B to that state k_{C0} if the bit indexed by i, j, and k is 0 or the state k_{C1} if the bit is 1.

Our bit gadget has Nondeterministically looked up each bit indexed by it's A and B states, Now, we must compare the bit we just retrieved to the C index via the state in the C column. The states k_{C0} and k_C transition changing the state k_C to the $0i$ state only when they represent the same k. The same is true for the state k_{C1} except C_k transitions to $1i$.



(a) ST Bit Gadget look up (b) Nondeterminstic Bit Gadget look up

Figure 2.7: (a.u) For a string S, where the first 3 bits are 001, the states 0_A and 0_B have $|S|^{\frac{1}{3}}$ transition rules changing the state 0_B to a state representing one of the first $|S|^{\frac{1}{3}}$ bits. The state is i_{C0} if the i^{th} bit is 0 or i_{C1} if the i^{th} bit is 1 (a.v) The state 0_{C0} and the state 0_C both represent the same C index so the 0_C state transition to the 0_s. (a.w) For all states not matching the index of 0_C, they transition to x_B, which can be seen as a blank B state. (a.x) The state 0_{Bu} transitions with the state x_B changing to 0_B resetting the bit gadget. (b.a) Once the state A_0 appears in the bit gadget it transitions with 0_B changing 0_B to $0'_B$. (b.b) The states $0'_B$ and 0_C Nondeterminstically look up bits with matching B and C indices. The state $0'_B$ transitions to look up state representing the bit retrieved and the bit's A index. The state 0_C transitions to a look up state representing the D index of the retrieved bit. (b.c) The look-up states transition with the states 0_A and 0_D, respectively. As with the Single-Transition construction these may pass or fail. (b.d) When both tests pass, they transition the D look up state to a symbol state that propagates out. (b.e) If a test fails, the states both go to blank states. (b.f) The blank states then reset using the states to their north.

If they both represent different k, then the state k_C goes to the state B_x. This is the error checking of our system. The B_x states transitions with the north state j_{Bu} above it transitioning B_x to j_B once again. This takes the bit gadget back to it's starting configuration and another look up can occur.

Theorem 2.3.4. For any binary string S with length $n > 0$, there exists a Single-Transition tile automata system Γ, that uniquely assembles an $(2n+2) \times 3$ assembly which represents S with $\mathcal{O}(n^{\frac{1}{3}})$ states.

2.3.3 General Nondeterministic Transitions

Using a similar method to the previous sections, we build length n strings using $\mathcal{O}(n^{\frac{1}{4}})$ states. We start by building a pattern of index states with bit gadgets of height 2 and width 4.

Overview Here, let $r = \lceil n^{\frac{1}{4}} \rceil$. We build index states in the same way as the Single-Transition system but instead starting from the D column. We have 4 sets of index states, A, B, C, D. The same

methods are used to control when the next section builds by transitioning the state $r-1_D$ to $r-1'_D$ when the current section is finished building.

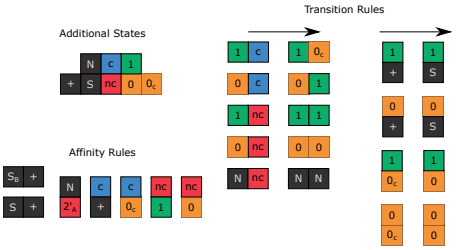
We use a similar look up method as the previous construction where we Nondeterministically retrieve a bit. However, since we are not restricting our rules to be a Single-Transition system, we may retrieve 2 indices in a single step. We include 2 sets of $\mathscr{O}(r)$ look up states, the A look up states and the D look up states. We also include Pass and Fail states $F_B, F_C, P_{A0}, P_{D0}, P_{A1}, P_{D1}$ along with the blank states B_x and C_x. We utilize the same method to build the north and south row.

Let $S(\alpha,\beta,\gamma,\delta)$ be the i^{th} bit of S where $i=\alpha r^3+\beta r^2+\gamma r+\delta$. The states β'_B and γ_C have r^2 transitions rules. The process of these transitions is outlined in Figure 2.7b. They transition from (β'_B,γ_C) to either $(\alpha_{A0},\delta_{D0})$ if $S(\alpha,\beta,\gamma,\delta)=0$, or $(\alpha_{A1},\delta_{D1})$ if $S(\alpha,\beta,\gamma,\delta)=1$. After both transitions have happened, we test if the indices match to the actual A and D indices. We include the transition rules (α_A,α_{A0}) to (α_A,P_{A0}) and (α_A,α_{A1}) to (α_A,P_{A1}). We refer to this as the bit gadget passing a test. The two states (P_{A0},P_{D0}) horizontally transition to $(P_{A0},0s)$. The $0s$ state then transitions the state δ_D to $0s$ as well as propagating the state to the right side of the assembly. If the compared indices are not equal, then the test fails and the look up states will transition to the fail states F_B or F_C. These fail states will transition with the states above them, resetting the bit gadget as in the previous system.

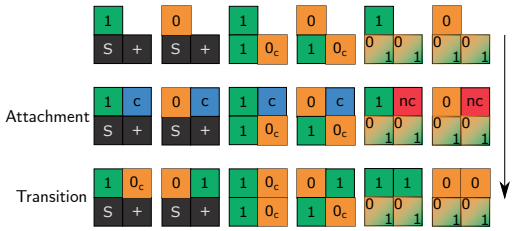
Theorem 2.3.5. For any binary string S with length $n>0$, there exists a tile automata system Γ, that uniquely assembles an $(2n+2)\times 4$ assembly which represents S with $\mathscr{O}(n^{\frac{1}{4}})$ states.

2.4 Rectangles

In this section, we will show how to use the previous constructions to build $\mathscr{O}(\log n)\times n$ rectangles. All of these constructions rely on using the previous results to encode and print a string then adding additional states and rules to build a counter.

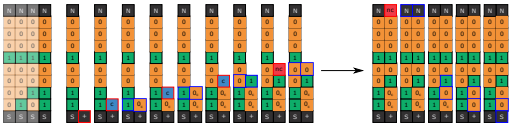


(a) New states and rules for a binary counter

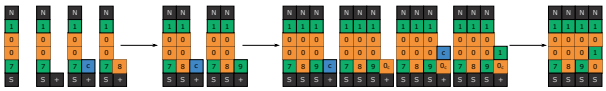


(b) Every case for the half adder.

Figure 2.8: (b) The 0/1 tile is not present in the system. It is used in the diagram to show that either a 0 tile or a 1 tile can take that place.



(a) Binary Counter



(b) Base-10 Counter

Figure 2.9: (a) The process of the binary counter. (b) A base-10 counter.

2.4.1 States

We choose a string and construct a system that will create that string, using the techniques shown in the previous section. We then add states to implement a binary counter that will count up from the initial string. The states of the system, seen in Figure 2.8a, have two purposes. The north and south states (N and S) are the bounds of the assembly. The plus, carry, and no carry states (+, c, and nc) forward the counting. The 1, 0, and 0 with a carry state make up the number. The counting states and the number states work together as half adders to compute bits of the number.

2.4.2 Transition Rules / Single Tile Half Adder

As the column grows, in order to complete computing the number, each new tile attached in the current column along with its west neighbor are used in a half adder configuration to compute the next bit. Figure 2.8b shows the various cases for this half adder.

When a bit is going to be computed, the first step is an attachment of a carry tile or a no-carry tile (c or nc). A carry tile is attached if the previous bit has a carry, indicated by a tile with a state of plus or 0 with a carry (+ or 0c). A no-carry tile is placed if the previous bit has no-carry, indicated

by a tile with a state of 0 or 1. Next, a transition needs to occur between the newly attached tile and its neighbor to the west. This transition step is the addition between the newly placed tile and the west neighbor. The neighbor does not change states, but the newly placed tile changes into a number state, 0 or 1, that either contains a carry or does not. This transition step completes the half adder cycle, and the next bit is ready to be computed.

2.4.3 Walls and Stopping

The computation of a column is complete when a no-carry tile is placed next to any tile with a north state. The transition rule changes the no-carry tile into a north state, preventing the column from growing any higher. The tiles in the column with a carry transition to remove the carry information, as it is no longer needed for computation. A tile with a carry changes states into a state without the carry. The next column can begin computation when the plus tile transitions into a south tile, thus allowing a new plus tile to be attached. The assembly stops growing to the right when the last column gets stuck in an unfinished state. This column, the stopping column, has carry information in every tile that is unable to transition. When a carry tile is placed next to a north tile, there is no transition rule to change the state of the carry tile, thus preventing any more growth to the right of the column.

Theorem 2.4.1. For all $n > 0$, there exists a Tile Automata system that uniquely assembles a $\mathcal{O}(\log n) \times n$ rectangle using,

- Deterministic Transition Rules and $\mathcal{O}(\log^{\frac{1}{2}} n)$ states.
- Single-Transition Transition Rules and $\Theta(\log^{\frac{1}{3}} n)$ states.
- Nondeterministic Transition Rules and $\Theta(\log^{\frac{1}{4}} n)$ states.

2.4.4 Arbitrary Bases

Here, we generalize the binary counter process for arbitrary bases. The basic functionality remains the same. The digits of the number are computed one at a time going up the column. If a

digit has a carry, then a carry tile attaches to the north, just like the binary counter. If a digit has no carry, then a no-carry tile is attached to the north. The half adder addition step still adds the newly placed carry or no-carry tile with the west neighbor to compute the next digit. This requires adding $\mathcal{O}(b)$ counter states to the system, where b is the base.

Theorem 2.4.2. For all $n > 0$, there exists a Deterministic Tile Automata system that uniquely assembles a $\mathcal{O}(\frac{\log n}{\log\log n}) \times n$ rectangle using $\Theta\left((\frac{\log n}{\log\log n})^{\frac{1}{2}}\right)$ states.

2.5 Squares

In this section we utilize the rectangle constructions to build $n \times n$ squares using the optimal number of states.

Let $n' = n - 4\lceil\frac{\log n}{\log\log n}\rceil - 2$, and Γ_0 be a determinstic Tile Automata system that builds a $n' \times (4\lceil\frac{\log n}{\log\log n}\rceil + 2)$ rectangle using the process described in Theorem 2.4.2. Let Γ_1 be a copy of Γ_0 with the affinity and transition rules rotated 90 degrees clockwise, and the state labels appended with the symbol "*1". This system will have distinct states from Γ_0, and will build an equivalent rectangle rotated 90 degrees clockwise. We create two more copies of Γ_0 (Γ_2 and Γ_3), and rotate them 180 and 270 degrees, respectively. We append the state labels of Γ_2 and Γ_3 in a similar way.

We utilize the four systems described above to build a hollow border consisting of the four rectangles, and then adding additional initial states which fill in this border, creating the $n \times n$ square.

We create Γ_n, starting with system Γ_0, and adding all the states, initial states, affinity rules, and transition rules from the other systems ($\Gamma_1, \Gamma_2, \Gamma_3$). The seed states of the other systems are added as initial states to Γ_n. We add a constant number of additional states and transition rules so that the completion of one rectangle allows for the "seeding" of the next.

Reseeding the Next Rectangle. To Γ_n we add transition rules such that once the first rectangle (originally built by Γ_0) has built to its final width, a tile on the rightmost column of the rectangle will transition to a new state pA. pA has affinity with the state $S_A * 1$, which originally

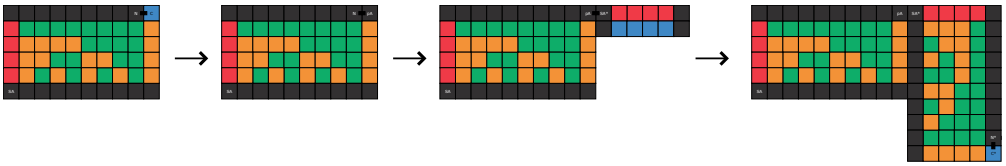


Figure 2.10: The transitions that take place after the first rectangle is built. The carry state transitions to a new state that allows a seed row for the second rectangle to begin growth

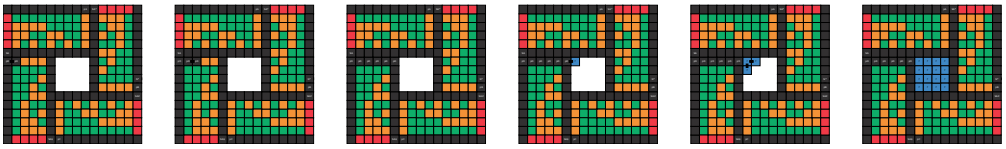


Figure 2.11: Once all 4 sides of the square build the *pD* state propagates to the center and allows the light blue tiles to fill in

was the seed state of Γ_1. This allows state $S_A * 1$ to attach to the right side of the rectangle, "seeding" Γ_1 and allowing the next rectangle to assemble (Figure 2.10). The same technique is used to seed Γ_2 and Γ_3.

Filler Tiles. When the construction of the final rectangle (of Γ_3) completes, transition rules propagate a state *pD* towards the center of the square (Figure 2.11). Additionally, we add an initial state *r*, which has affinity with itself in every orientation, as will as with state *pD* on its west side. This allows the center of the square to be filled with tiles.

Theorem 2.5.1. For all $n > 0$, there exists a Tile Automata system that uniquely assembles an $n \times n$ square with,

- Deterministic transition rules and $\Theta\left(\left(\frac{\log n}{\log\log n}\right)^{\frac{1}{2}}\right)$ states.
- Single-Transition rules and $\Theta(\log^{\frac{1}{3}} n)$ states.
- Nondeterministic transition rules and $\Theta(\log^{\frac{1}{4}} n)$ states.

2.6 Future Work

This paper showed optimal bounds for uniquely building $n \times n$ squares in three variants of seeded Tile Automata without cooperative binding. En route, we proved upper bounds for

constructing strings and rectangles. Serving as a preliminary investigation into constructing shapes in this model. This leaves many open questions:

- As shown in [9], even 1D Tile Automata systems can perform Turing computation. This behavior may imply interesting results for constructing $1 \times n$ lines. We conjecture, it is possible to achieve the optimal bound of $\Theta((\frac{\log n}{\log \log n})^{\frac{1}{2}})$ with deterministic rules.

- Our rectangles had a height bounded by $\mathcal{O}(\frac{\log n}{\log \log n})$, and none fell below the $k < \frac{\log n}{\log \log n}$ [2] bound for a thin rectangle. In Tile Automata without cooperative binding, is it possible to optimally construct $k \times n$ thin rectangles?

- We allow transition rules between non-bonded tiles. Can the same results be achieved with the restriction that a transition rule can only exist between two tiles if they share an affinity in the same direction?

- While we show optimal bounds can be achieved without cooperative binding, can we simulate so-called zig-zag aTAM systems? These are a restricted version of the cooperative aTAM that is capable of Turing computation.

- We show efficient bounds for constructing strings in Tile Automata. Given the power of the model, it should be possible to build algorithmically defined shapes such as in [39] by printing Komolgorov optimal strings and inputting them to a Turing machine.

CHAPTER III

PATTERNS

3.1 Model and Definitions

Here, we quickly review basic definitions and provide full formal definitions of these concepts in Appendix **??**.

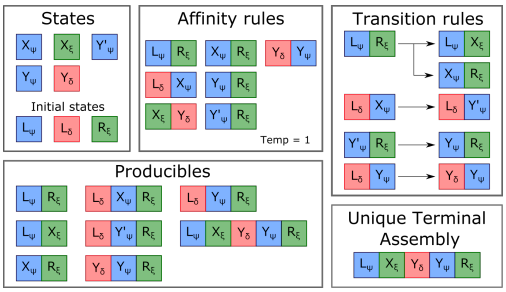
States, Tiles, and Assemblies. A *tile* is a unit square centered along the square lattice. Each tile is assigned a *state* from the alphabet Σ. An *assembly* is a collection of adjacent tiles at disjoint locations.

Affinity and Transition Rules. Systems consist of local affinities between states denoting the direction and strength of attraction between adjacent monomer tiles in those states, and a temperature or threshold defining the minimum attraction strength needed to bind. Our systems all use a threshold of 1. A set of local state-change rules, or transition rules, are included for pairs of two adjacent states.

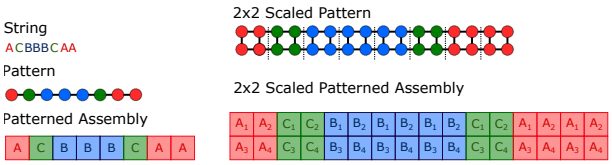
Breakable, Combinable, Transitionable Assemblies. For two assemblies to combine, the border of where they meet must have the total affinity strength sum to at least the temperature of the system. An assembly may also break apart into two assemblies when the total affinity strength along the border of where each assembly meet sum to less than the temperature of the system. Further, existing assemblies may change states based on the transition rules.

3.1.1 Tile Automata model (TA)

A *Tile Automata system* is a 5-tuple $(\Sigma, \Pi, \Lambda, \Delta, \tau)$ where Σ is an alphabet of state types, Π is an affinity function, Λ is a set of initial assemblies with each tile assigned a state from Σ, Δ is a



(a) A Tile Automata System



(b) A patterned assembly and scaled patterned assembly

Figure 3.1: (a) An example of a Tile Automata system Γ. Recursively applying the transition rules and affinity functions to the initial assemblies of a system yields a set of producible assemblies. Any producibles that cannot combine with, break into, or transition to another assembly are considered to be terminal. Note that none of the transition rules allow states to change color. (b) For the pattern problem, an input *string* defining the *pattern* and a *patterned assembly* representing it. Also shown is the 2×2 *scaled pattern* and the corresponding 2×2 *scaled patterned assembly*.

set of transition rules for states in Σ, and $\tau \in \mathbb{N}$ is the *stability threshold*. When the affinity function and state types are implied, let (Λ, Δ, τ) denote a tile automata system. An example Tile Automata system can be seen in Figure **??**. Please see Appendix **??**.

Definition 3.1.1 (Tile Automata Producibility)**.** For a given Tile Automata system $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, \tau)$, the set of *producible assemblies* of Γ, denoted $\texttt{PROD}_\Gamma$, is defined recursively:

- (Base) $\Lambda \subseteq \texttt{PROD}_\Gamma$
- (Recursion) Any of the following:
  - (Combinations) For any $A, B \in \texttt{PROD}_\Gamma$ s.t. A and B are τ-combinable into C, then $C \in \texttt{PROD}_\Gamma$.
  - (Breaks) For any $C \in \texttt{PROD}_\Gamma$ s.t. C is τ-breakable into A and B, then $A, B \in \texttt{PROD}_\Gamma$.
  - (Transitions) For any $A \in \texttt{PROD}_\Gamma$ s.t. A is transitionable into B (with respect to Δ), then $B \in \texttt{PROD}_\Gamma$.

For a system $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, \tau)$, we say $A \to_1^\Gamma B$ for assemblies A and B if A is τ-combinable with some producible assembly to form B, if A is transitionable into B (with respect to Δ), if A is τ-breakable into assembly B and some other assembly, or if $A = B$. Intuitively this means that A

may grow into assembly B through one or fewer combinations, transitions, and breaks. We define the relation $\rightarrow^{\Gamma}$ to be the transitive closure of $\rightarrow_1^{\Gamma}$, i.e., $A \rightarrow^{\Gamma} B$ means that A may grow into B through a sequence of combinations, transitions, and/or breaks.

Definition 3.1.2 (Production Graph)**.** The production graph of a Tile Automata system Γ is a directed graph where each vertex corresponds to an assembly in $\texttt{PROD}_{\Gamma}$ and there exists a directed edge between assemblies A and B if $A \rightarrow^{\Gamma} B$.

Definition 3.1.3 (Terminal Assemblies)**.** A producible assembly A of a Tile Automata system $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, \tau)$ is *terminal* provided A is not τ-combinable with any producible assembly of Γ, A is not τ-breakable, and A is not transitionable to any producible assembly of Γ. Let $\texttt{TERM}_{\Gamma} \subseteq \texttt{PROD}_{\Gamma}$ denote the set of producible assemblies of Γ that are terminal.

Definition 3.1.4 (Bounded)**.** A tile automata system Γ is *bounded* if and only if there exists a $k \in \mathbb{Z}_{>0}$ such that for all $A \in \texttt{PROD}_{\Gamma}$, $|A| < k$.

Definition 3.1.5 (Unique Assembly)**.** A TA system Γ *uniquely produces* an assembly A if

- A is the only assembly in $\texttt{TERM}_{\Gamma}$,
- for all $B \in \texttt{PROD}_{\Gamma}$, $B \rightarrow^{\Gamma} A$, and
- Γ is bounded.

When there does not exist a pair of assemblies $B, C \in \texttt{PROD}_{\Gamma}$, such that $B \rightarrow^{\Gamma} C \rightarrow^{\Gamma} B$, we refer to it as *cycle-free*. Cycles are only possible when detachment is allowed.

3.1.2 Restrictions

Based on hierarchies that have been studied in Cellular Automata [34], we also restrict our TA systems. Unique Assembly and directedness captures the idea of convergence from Cellular Automata where the system is required to eventually end at some configuration. Allowing or disallowing cycles (cycle-free) is similar to the concept of bounded change. While we do not focus

on this, we note the only systems *without* this property are the nondeterministic transition rule results based on the efficient string constructions from [3].

The main restrictions considered here are:

- **Directed.** If a system Γ uniquely produces some assembly A allowing cycles we say Γ is directed. This concept is well-defined in models such as the aTAM.

- **Freezing.** We say a system is freezing if a tile only ever visits each state at most once. The concept of freezing has been studied in Cellular Automata [25] where is it defined as having a fixed order on the states. However, we note that these definitions are equivalent for deterministic systems based on the state transitions.

- **Affinity Strengthening.** A TA system where, whenever a state a transitions to state b, b must have at least the same affinity rules as a. This prevents detachment.

- **Determinism.** A deterministic system in TA has only one possible production step at a time, whether that be an attachment or a state transition. A nondeterministic system may have many possible production steps where any choice may be taken.

- **Single-Transition System.** We restrict our TA system to only use single-transition rules. Thus, for each transition rule, only one of the states may change.

3.1.3 Colors and Patterns

In this paper, we augment the Tile Automata model with the concept of a tile's color based on the current state. For a set of color labels C, this is a partition of the states into $|C|$ sets. We only consider constant sized C. Thus, the *color* of tile t is the partition the tile's state is in, and is denoted as $c(t)$.

Definition 3.1.6 (Pattern)**.** A pattern P over a set of colors C is a partial mapping of $\mathbb{Z}^2$ to elements in C. Let $P(z)$ be the color at $z \in \mathbb{Z}$. A scaled pattern P^{hw} is the pattern where each pixel is replaced by a $h \times w$ rectangle of pixels.

Definition 3.1.7 (Patterned Assemblies)**.** We say a positioned assembly A' represents a pattern P if for each tile $t \in A'$, $c(t) = P(L(t))$ and $dom(A') = dom(P)$. We say a positioned assembly B' represents a pattern P at scale $h \times w$ if it represents the scaled pattern P^{hw}.

A system Γ uniquely assembles a pattern P if it uniquely assembles an assembly A, such that A contains a positioned assembly that represents P.

3.1.4 Algorithmic Hierarchy

We measure the efficiency of building a pattern P using three algorithmic complexity values related to levels of the Chomsky Hierarchy.

- **Kolmogorov Complexity (K_P).** The Kolmogrov complexity the size of the smallest binary string that when input to some fixed universal Turing machine M_K it outputs the prints the pattern P. Note that K_P may differed based on the choice of M_K however it has been shown the choice only changes an additive constant.

- **Space Bounded Kolmogorov Complexity (KS_P).** The space bounded Kolmogorov complexity for some function $f(|P|)$, KS_P is the smallest binary string that when input to M_K is P is output in $f(|P|)$ space. Time and Space bounded Kolmogorov Complexity is explored in [29].

- **Context-Free (CF_P).** We denote the size of the smallest Context-Free Grammar as CF_P. The size of the grammar is the total number of symbols on the right hand side of the rules. We provide a formal definition in Section 3.4 and consider the same class of grammars as [17].

3.1.5 General Turing Machine

3.2 Optimal Patterns in Tile Automata

In this section we show that general Tile Automata can obtain Kolmogorov optimal state complexity at 1×1 scale. These first results are achieved by applying the efficient binary string construction from [3], and allowing the additional tiles used by the assembly to fall off, thus leaving

only the string. We can then utilize the Turing machine from [9] to simulate a universal Turing Machine. The Turing Machine in [9] was designed to accept/reject an input, so we modify the Turing Machine to print P on the tape and halt.

Lemma 3.2.1. For any binary pattern X there exists an affinity strengthening Tile Automata system that uniquely constructs an assembly representing X at scale,

- 4×2 with $\mathscr{O}(|X|^{\frac{1}{4}})$ states,
- 3×2 with $\mathscr{O}(|X|^{\frac{1}{3}})$ states using single-transition rules, and
- 2×1 with $\mathscr{O}(|X|^{\frac{1}{2}})$ states using deterministic single-transition rules and is cycle free.

Proof. These constructions are provided in [3] which shows that there exists a method to encode the bits of a string in the transition rules of the system. Each construction takes advantage of a feature not available in the stricter class of systems. The model shown in this paper however does have seeded growth but a simple extension shows this works with 2-handed production. □

Theorem 3.2.2. For any pattern P, there exists a Tile Automata system Γ that uniquely assembles P with $\Theta(K_P^{\frac{1}{4}})$ states at 1×1 scale.

Proof. Given a pattern P, we first consider a Turing machine M that will print P. Using the process described in [9], we create a system $\Gamma_M = (\Sigma, \Pi, \Lambda, \Delta, \tau)$ that simulates M. When M has completed printing P, the buffer states B_L and B_R need to detach. We take Σ and create a copy Σ_{SR} which we modify by removing the accept/reject states in favor of *final states*. For every state $\rho \in \Sigma_{SR}$ where ρ composes P, we create $\rho_F \in \Sigma_{SR}$ with affinity only for every other final state. Starting with the rightmost tile that composes P, we add transition rules that will transition each tile with state ρ into their final state equivalent ρ_F. Since these final states have no affinity with the buffer states, tiles with those buffer states, and any other state not considered a final state, will detach from the assembly. This detaching process begins with a transition rule between B_R and the rightmost tile with state ρ, turning ρ into ρ_F.

From Lemma 3.2.1, we encode Γ_M in a binary string $b(\Gamma_M)$ and use $b(\Gamma_M)$ to construct system Γ_S that uses $\Theta(K_P^{\frac{1}{4}})$ to assemble $b(\Gamma_M)$. [44] states there exists a universal Turing machine that uses linear space in the amount of space used by the machine being simulated. Γ will simulate a universal Turing machine with Γ_S being used to construct the input into Γ, giving us a system that uniquely assembles P with $\Theta(K_P^{\frac{1}{4}})$ states and 1×1 scale. □

3.2.1 Deterministic Single Transition Turing Machine

The Turing machine from [9] utilizes transition rules that change both tiles in the same step. While [11] shows a way to simulate double rules with single rules, we present a slight modification to the Turing machine construction to make it utilize single rules.

Lemma 3.2.3. For any pattern P, there exists a Tile Automata system Γ with deterministic single-transition rules that uniquely assembles P with $\mathscr{O}(K_P)$ states and 1×1 scale. This system is cycle free.

Proof. We create a Turing machine M that will print P. Using Turing machine M, we use the process described in [9] to create a system $\Gamma_D = (\Sigma, \Pi, \Lambda, \Delta, \tau)$ that simulates M utilizing double-transition rules. We then modify Σ, Δ, and Π into single-transition rule versions Σ_{SR}, Δ_{SR}, and Π_{SR} as follows.

Σ_{SR} and Π_{SR} will initially be a copy of Σ and Π respectively, while Δ_{SR} is populated with every single-transition rule in Δ. For every double-transition rule $\delta = (\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}, d) \in \Delta$, we create an additional state $\omega \in \Sigma_{SR}$. The affinity strength of ω using Π_{SR} will be equal to the affinity strength of D using Π for all directions. We take δ and create 3 transition rules $\delta_{S1}, \delta_{S2}, \delta_{S3} \in \Delta_S$ defined below.

- $\delta_{S1} = (\mathsf{A}, \mathsf{B}, \mathsf{A}, \omega, d)$
- $\delta_{S2} = (\mathsf{A}, \omega, \mathsf{C}, \omega, d)$
- $\delta_{S3} = (\mathsf{C}, \omega, \mathsf{C}, \mathsf{D}, d)$

We use the *final states* described in the proof of Theorem 3.2.2 to modify Σ_{SR} in order to detach the buffer states. Using our modifications, we create a Tile Automata system $\Gamma = (\Sigma_{SR}, \Pi_{SR}, \Lambda, \Delta_{SR}, \tau)$ with deterministic single-transition rules that uniquely assembles P with $\mathscr{O}(K_P)$ states and 1×1 scale. □

Using Lemma 3.2.1we can encode the input to a universal Turing machine with square root the number of states with deterministic single transition rules.

Theorem 3.2.4. For any pattern P, there exists a Tile Automata system Γ with deterministic single transition rules that uniquely assembles P with $\mathscr{O}(K_P^{\frac{1}{2}})$ states and 1×1 scale. This system is cycle free.

Proof. We make some modifications to the process used in the proof of Theorem 3.2.2 to satisfy the deterministic single-transition rules. We create Γ_M using the method described in the proof of Lemma 3.2.3 and encode the system in a binary string $b(\Gamma_M)$. Γ_S is created using $b(\Gamma_M)$ which will use $\mathscr{O}(K_P^{\frac{1}{2}})$ as shown in Lemma 3.2.1. Γ will simulate a universal Turing machine that uses the assembly built by Γ_S, giving us a system that uniquely assembles P with $\mathscr{O}(K_P^{\frac{1}{2}})$ states and 1×1 scale. □

Other methods for non-deterministic rules and with single and double rules give the following.

Theorem 3.2.5. For any pattern P, there exists a Tile Automata system Γ with single transition rules that uniquely assembles P with $\Theta(K_P^{\frac{1}{3}})$ states and 1×1 scale.

Proof. A deterministic single-rule TA system Γ_M can be constructed according to Lemma 3.2.3, and using an encoding $b(\Gamma_M)$, we make Γ_S which uses $\Theta(K_P^{\frac{1}{3}})$ states using Lemma 3.2.1 □

3.2.2 Freezing with Detachment

We do not directly consider Freezing and allowing detachment since the results of [12] shown that any non-freezing system can be simulated by a freezing system by replacing tiles. Also

shown in the full version of [9] it was shown freezing Tile Automata with only height 2 assemblies can simulate a general Turing machine. The assembly can then fall apart to achieve 1×1 scale.

3.2.3 Affinity Strengthening

3.3 Affinity Strengthening

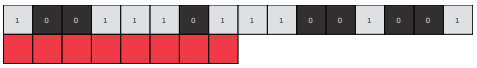
As shown in [9], Affinity Strengthening Tile Automata (ASTA) is capable of simulating Linear Bounded Automata (LBA) and that verification in ASTA is PSPACE-Complete. Thus, it makes sense to view this version of the model as the spaced-bounded version of Tile Automata, similar in power to LBAs or Context Sensitive Grammars. We select space-bounded Kolmogorov complexity as our method of bounding the state complexity since we can encode a string and simulate a Turing machine as in the previous section to get an upper bound. The concept of bounded Kolmogorov Complexity was explored in [29]. For these results, we consider building scaled patterns in which each pixel of the pattern is expanded to a $s \times O(1)$ box of pixels. Another way to view this upper bound is that for any algorithm α that outputs P in $f(|P|)$ space, we may construct an assembly representing P of size $\mathscr{O}(f(|P|)$, in $\mathscr{O}(|\alpha|)^{\frac{1}{4}}$ states, where $|\alpha|$ is the number of bits describing α for general Tile Automata. Similar bounds are shown for the other restrictions. It is interesting to point out that with a large enough scale factor we achieve Kolmogorov optimal bounds, including optimal scaled shape constructions as in [39].

3.3.1 Space Bounded Kolmogorov Complexity

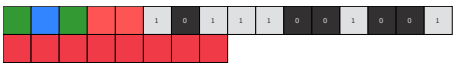
Definition 3.3.1 (Space Bounded Kolmogorov Complexity)**.** Given a pattern P, and a function $f : \mathbb{N} \to \mathbb{N}$ that outputs the space used by a Turing machine, let $KS_P(f(|P|))$ be the length of the smallest string that, when input to a universal Turing machine M_K, halts with the pattern P on the tape in $f(|P|)$ space.

It was stated in [29] that there exists some optimal Turing machine, which we call M_K, that incurs only a constant multiplicative factor increase in the space used. We note for two space bounds $f(|P|)$ and $g(|P|)$, the value $KS_P(g(|P|)) \leq KS_P(f(|P|))$ as using more space allows for

(a) Assembly representing input string.



(b) Universal Turing machine running on input.



(c) The pattern is output on the tape.

(d) The pattern scales outward to fill the assembly.

Figure 3.2: (a) It is possible to build assemblies representing binary strings with an efficient number of states. (b) We can then run a universal Turing machine on the input increasing the length of the assembly as needed. (c) The Turing machine will halt with the pattern output on the tape. (d) The pattern will then scale out to fill the assembly.

more efficient computing of all pattern P, with $|P| < c$ for some constant c.

3.3.2 Construction

Figure 3.2a shows a sketch of the assembly for deterministic Tile Automata using the string constructions from [3] shown in Lemma 3.2.1. The single rule Turing machine can be modified to never break apart and only increase the tape length, similar to the PSPACE-hard reduction from [9]. Figure 3.2b shows an example Turing machine being run where the tape length is increased.

Once the pattern has been printed or assembled (Figure 3.2c), there are additional tiles in the assembly to deal with. However, since we cannot detach tiles, we scale the pattern. The first step is to expand the length of pattern. If we use s tape cells to print a pattern $|P|$, we scale each point in the pattern by $c \cdot |P|$. This is done with a simple algorithm implemented in the transition rules. Create a token state that starts at the leftmost state after the string is printed. Go to each 'pixel' and tell it expand once after first signaling the neighboring cells to move right (to prevent overwriting). We do this for each pixel in the pattern, push the other states, increase pixel size. The system repeats this process until all pixels of the pattern are fully expanded, and then they transition the tiles below them, which results in the patterned assembly of Figure 3.2d.

Theorem 3.3.2. For any pattern P, scale factor $s > 0$, there exists an Affinity Strengthening Tile Automata system Γ with deterministic single transition rules that uniquely assembles P with $\mathcal{O}(KS_P(s|P|)^{\frac{1}{2}})$ states and $s \times 2$ scale. This system is cycle free.

Proof. Let X be the string that when input to M, P is written to the tape in $s|P|$ space. Using the binary string building results from Lemma 3.2.1 we can encode X in $\mathcal{O}(|X|^{\frac{1}{2}})$ states. Then we run M using the single transition rule Turing machine described in the proof of Lemma 3.2.3. This will run and leave the pattern P on the tape states. Consider a second Turing machine M_{INC} scales up the pattern to fill the width of the tape. Each pixel is increased by the same amount. The states then copy the color to the state below it as well. This can be done in a constant number of states. The amount that each pattern scales by is $\frac{s|P|}{|P|} = |P| \cdot (s-1)$. □

Theorem 3.3.3. For any pattern P, scale factor $s > 0$, there exists an Affinity Strengthening Tile Automata system Γ with single-transition rules that uniquely assembles P with $\mathcal{O}(KS_P(s|P|)^{\frac{1}{3}})$ states and $s \times 3$ scale.

Proof. Again using the Single-Transition rule Turing machine from the proof of Lemma 3.2.3 and the string building result from Lemma 3.2.1, we can construct the input to the universal Turing machine M_K. The pattern P can be output in $s|P|$ space. We then scale up the pattern to fill the assembly. □

Theorem 3.3.4. For any pattern P, scale factor $s > 0$, there exists an Affinity Strengthening Tile Automata system Γ that uniquely assembles P with $\mathcal{O}(KS_P(s|P|)^{\frac{1}{4}})$ states and $s \times 4$ scale.

Proof. Lastly using the same method from Lemma 3.2.1 we can encode the input to the universal Turing machine in $|X|^{\frac{1}{4}}$ where $|X|$ is the length of the string. This results in an assembly of height 4 as resulting assembly will be of dimensions $|X| \times 4$. The string X can then be input to the Turing machine to print the pattern than scale up. □

3.4 Freezing Affinity Strengthening

In this section, we design a Freezing Tile Automata system to build a pattern P based on a context-free grammar (CFG) that encodes P. This construction draws inspiration from pattern building in staged self-assembly [17] and line building in Tile Automata and the Signal Tile

Model [9, 35]. An example CFG is shown in Figure 3.3, along with the corresponding TA system in Figure **??**. In addition to the freezing and affinity strengthening constraints, this result achieves the feature that tiles never undergo a change in their color throughout the assembly process. We denote rules that adhere to this constraint as *color-locked* rules.

3.4.1 Context-Free Grammars

A **context-free grammar (CFG)** is a set of recursive rules used to generate patterns of strings in a given language. A CFG is defined as a quadruple $G = (V, \Upsilon, R, S)$. V represents a finite set of non-terminal symbols and Υ is a finite set of terminal symbols. The symbol R is the set of production rules and S is a special variable in V called the start symbol. Production rules R of CFGs are in the form $A \to BC|a$, with V in the left-hand side and V and/or Υ on the right- hand side. A CFG derives a string through recursively replacing nonterminal symbols with terminal and non-terminal symbols based on its production rules.

Definition 3.4.1 (Minimum Context Free Grammars)**.** We define the size of a grammar G as the number of symbols in the right hand side rules. Let CF_P be the size of the smallest CFG that produces the singleton language $|P|$.

Restricted Context-Free Grammars (RCFG). In this work, we focus on the CFG class used in [17] which they name Restricted CFGs. These restricted grammars produce a singleton language, $|L(G)| = 1$ and thus are deterministic. This is the same concept of Context-Free Straight Line grammars from [7]. Each RCFG production rule R contains two symbols on its right-hand side. We can convert any other deterministic CFG to this form with only a constant factor size increase.

Figure 3.3 presents an example RCFG G and its parse tree that derives a pattern of symbols P, $\xi\xi\delta\delta\delta\psi$. The parse tree shows how internal nodes are non-terminal symbols and leaf nodes contain a terminal symbol whose in-order traversal derives the output string. Notice that since RCFG G is deterministic, each non-terminal symbol $N \in V$ has a unique subpattern $g(N)$ that is defined by taking N to be the start symbol S and applying the production rules. Here, the language

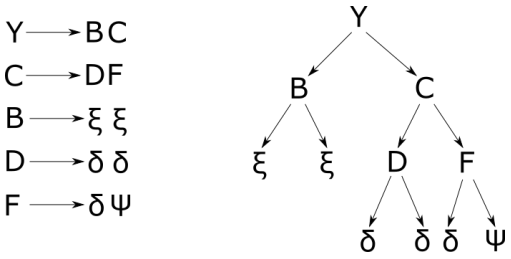


Figure 3.3: A restricted context-free grammar (RCFG) G and its corresponding parse tree that produces a pattern P, $\xi\xi\delta\delta\delta\psi$. This is a deterministic grammar, producing only pattern P.

or output pattern P of G can be denoted by $L(G) = g(S)$.

3.4.2 1D Patterned Assembly Construction

We describe our method of simulating a Restricted CFG G with Tile Automata to build a 1D patterned assembly that represents the pattern P derived from G.

Initial Tiles and Producibles. This Tile Automata system, Γ_G, begins with creating its initial tiles from the unique terminal symbols, Υ, in RCFG G. In Figure 3.3, the output pattern P derived from G has three unique terminal symbols ξ, δ, and ψ. Each unique Υ in G is mapped to a distinct color and remains locked to the symbol throughout the construction. From G's production rule parse tree, internal nodes have two child nodes consisting of two similar or different terminal symbols, Υ. Depending on the placement of the terminal symbols, the initial tiles are designated as L for left-hand side or R for right-hand side. Figure 3.4a depicts that an initial tile consists of an Υ symbol with its distinct color in an L or R state.

Following G's parse tree, the initial tiles can combine to build Γ_G's first set of producible assemblies. Grammar G's production rules can be encoded into system Γ_G by providing the affinity rules. If two terminal symbols in G connect to the same internal node in its parse tree, the initial tiles in Γ_G that represent the symbols combine to form a producible. The first set of producibles cannot bind to any other tile because they are capped with L and R states, which we denote as *captiles*, and thus are stopped from growing, shown in Figure 3.4b. Note that these first producibles are subpatterns of P.

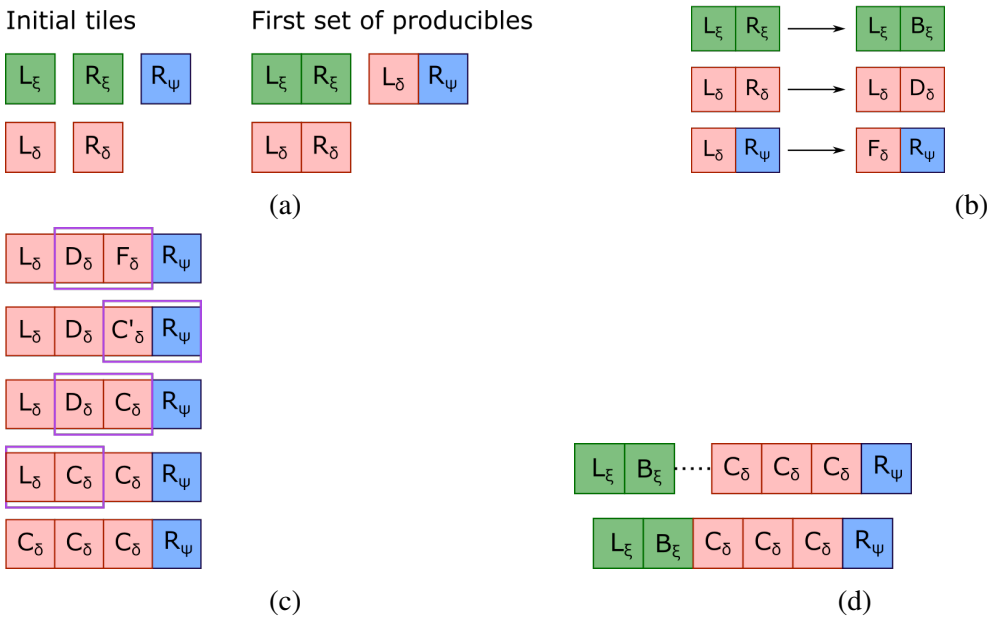


Figure 3.4: Tile Automata system, Γ_G, assembling a 1D patterned assembly that represents the pattern P produced by the RCFG G shown in Figure 3.3. (a) Γ_G contains initial tiles from the unique terminal symbols of G. Grammar G's production rules are encoded in Γ_G as affinity rules, allowing initial tiles to form the first set of producibles. (b) Following G's rules, Γ_G's color-locked, one-sided transition rules are applied to the first set of producibles. (c) Subpattern assembly $L_\delta D_\delta F_\delta R_\psi$ transitions tiles towards captile R, marking visited tiles. Once the transitions reach captile R, we transition to the left of the subassembly to C_δ tiles, removing the marks along the way. (d) RCFG G production rule $Y \rightarrow BC$, directs Γ_G to combine B and C subassemblies to build the terminal patterned line assembly, representing pattern P from grammar G.

Uncapping Producibles. RCFG G production rules tell Γ_G how the first producible assemblies will combine to form larger subpatterns of P and ultimately represent the final patterned line assembly. In Γ_G, our first set of producibles are composed of L and R captiles. For these producibles to combine with each other, we apply one-sided, color-locked transition rules to uncap each producible, opening their left or right-hand side depending on the nonterminal symbols placement in grammar G's production rules. For example, in Figure 3.3 nonterminal C is composed of a D on the left-hand side and F on the right-hand side. In Figure 3.2b, the producible $L_\delta R_\psi$ represents G's terminal symbols $\delta\psi$ as well as nonterminal F. Because F sticks to D's right side, a one-sided transition rule is applied to producible $L_\delta R_\psi$ changing only the pink tile L_δ to a new

tile F_δ, forming next producible $F_\delta R_\psi$. Here, the color-locked restriction in Γ_G applies because the new tile F_δ retains its color (pink) that is designated to the terminal symbol δ of P from G. This producible $F_\delta R_\psi$ is considered a right-handed subassembly because it is uncapped on its left side, allowing it to attach to the right-hand side of the producible that represents nonterminal D. The rest of Γ_G's first producibles transition according to G's production rules as shown in Figure 3.4b.

Transition Walk. Γ_G recursively applies G's production rules to build the other subassemblies needed to represent pattern P. Grammar G's production rule $C \to DF$ tells Γ_G that there is affinity between D and F, directing producibles $L_\delta D_\delta$ and $F_\delta R_\psi$ to combine and form a new subpattern assembly $\delta\delta\delta\psi$ of P, shown at the top of Figure 3.4c. In Lemma 3.4.3, we show how every nonterminal in G is represented as a subpattern assembly produced by Γ_G. Subpattern assembly $L_\delta D_\delta F_\delta R_\psi$, represents nonterminal C from G and is capped with captiles L and R. From G's production rules in Figure 3.3, nonterminal symbol Y is composed of B on the left-hand side and C on right-hand side. To uncap the left side of subpattern $L_\delta D_\delta F_\delta R_\psi$, a series of one-sided, color-locked transition rules are applied to turn each tile into a C_δ tile making the subassembly uniform, depicted in Figure 3.4c. The adjacent tiles that have transition rules between them are outlined in purple, with the resulting tiles shown in the subassembly below it.

We apply the method of "walking" across 1D assemblies from [9] to uncap left or right sides of subassemblies. Subpattern assembly $L_\delta D_\delta F_\delta R_\psi$ must have an opened left side to attach to subassembly B, so we first transition tiles towards the right side, marking visited tiles with a prime notation. Once the transitions reach captile R, we begin to transition to the left of the subassembly to C_δ tiles, removing the prime notations along the way. As shown in Figure 3.4c, once producibles D and F combine, a one-sided, color-locked transition rule applies changing the F_δ tile for a temporary C'_δ tile, where the prime marks the tile as visited. Next, the adjacent C'_δ and R_ψ tiles transition to remove the prime from the C'_δ tile, producing subpattern $L_\delta D_\delta C_\delta R_\psi$. Another transition is applied between adjacent tiles $D_\delta C_\delta$ to form the fourth subassembly in Figure 3.4c. Finally, one more transition occurs between $L_\delta C_\delta$ to produce subpattern $C_\delta C_\delta C_\delta R_\psi$.

Patterned Line Assembly. Figure 3.4d depicts the subpattern assemblies created by Γ_G that represent nonterminal symbols B and C. According to the affinity rules of Γ_G, subassemblies B and C combine to form terminal assembly Y. Subassemblies for B and C attach and terminal assembly Y is constructed and capped with captiles L and R on its sides. This new terminal assembly Y represents G's pattern P, with each distinct colored tile representing unique terminal symbols of pattern P.

Definition 3.4.2 (Nonterminal Pattern)**.** For a nonterminal $N \in V$, let $g(N)$ be a substring derived when N is the start symbol of grammar G.

Lemma 3.4.3. Each producible assembly in Γ_G, created from a RCFG $G = (V, \Upsilon, R, S)$ represents a subpattern $g(N)$ for some symbol N in $V \bigcup \Upsilon$.

Proof. We will prove by induction that any producible assembly B represents a subpattern $g(N)$ for some symbol N in $V \bigcup \Upsilon$.

For the base case, if B is an initial tile, then B represents some terminal symbol $N \in \Upsilon$. For the inductive step, if B is a larger assembly, then we show B represents a non-terminal $N \in V$. We define the following two recursive cases. B is built from combining subassemblies C and D, we can assume these assemblies represent symbols N_C and N_D respectively. We know from how we defined our affinity rules if C and D can combine then there is some rule $N \to N_C N_D$. Then B represents the pattern $g(N) = g(N_C) \oplus g(N_D)$. B is producible via transition from an assembly C, B must represent the same subpattern as C since the transition rules do not change the color. □

Theorem 3.4.4. For any pattern P, there exists a Freezing Tile Automata system Γ with deterministic single transition rules that uniquely assembles P with $\mathcal{O}(CF_P)$ states and 1×1 scale. This system is cycle-free and transition rules do not change the color of tiles.

Proof. By definition, there exists a CFG G that produces P with $|G| = CF_P$. We construct the system Γ_G. From Lemma 3.4.3, each producible assembly B must represent a subpattern $g(N)$ for

some symbol N. The only terminal of Γ is the assembly representing the start symbol S since all other assemblies either can attach to another assembly or can transition. □

3.5 Future Work

A possible extension to our constructions is general shape constructors as in [39] to produce general shapes at optimal state complexity. However, the ability to perform computation in 1D can help us bound the scale factor more reasonably. Based on Space Bounded Kolmogrov Complexity, lower bounds for bounded scale shapes in other assembly models may be possible. Time Bounded Kolmogorov Complexity is more studied in works such as [4, 8].

Closing the lower and upper bound gap for deterministic transition rules might be possible using base conversations, as we can encode general base strings using the string building results. For affinity strengthening, a stronger lower bound for the case of seeded Tile Automata should be possible since reading the output of a system might require less space. Are there more efficient algorithms for the original model to raise the lower bound?

The system to simulate a CFG is very restricted: it is freezing, one uses all 1D assemblies, and tiles never change color. Is there a limited version of TA that is equivalent to CFGs? A similar result was shown in the one-dimensional Staged Assembly Model when only one assembly is produced in each bin [17]. Can the 2D upper bounds be improved with scaling as in [42]? This allows for encoding CFG symbols with geometry rather than states. Also introduced in [42] are Polyomino Context-Free Grammars, which generalizes CFGs to define patterned polyominos in 2D rather than strings. This extension seems to be possible in Freezing Affinity Strengthening Tile Automata and would help us further compare passive tiles being mixed in a certain order vs. programming the tiles themselves to perform actions.

BIBLIOGRAPHY

[1] L. ADLEMAN, Q. CHENG, A. GOEL, AND M.-D. HUANG, *Running time and program size for self-assembled squares*, in Proceedings of the thirty-third annual ACM symposium on Theory of computing, 2001, pp. 740–748.

[2] G. AGGARWAL, Q. CHENG, M. H. GOLDWASSER, M.-Y. KAO, P. M. DE ESPANES, AND R. T. SCHWELLER, *Complexities for generalized models of self-assembly*, SIAM Journal on Computing, 34 (2005), pp. 1493–1515.

[3] R. M. ALANIZ, D. CABALLERO, S. C. CIRLOS, T. GOMEZ, E. GRIZZELL, A. RODRIGUEZ, R. SCHWELLER, A. TENORIO, AND T. WYLIE, *Building squares with optimal state complexity in restricted active self-assembly*, in Proceedings of the Symposium on Algorithmic Foundations of Dynamic Networks, SAND'22, 2022. to appear.

[4] E. ALLENDER, M. KOUCKỲ, D. RONNEBURGER, AND S. ROY, *The pervasive reach of resource-bounded kolmogorov complexity in computational complexity theory*, Journal of Computer and System Sciences, 77 (2011), pp. 14–40.

[5] J. C. ALUMBAUGH, J. J. DAYMUDE, E. D. DEMAINE, M. J. PATITZ, AND A. W. RICHA, *Simulation of programmable matter systems using active tile-based self-assembly*, in DNA Computing and Molecular Programming, C. Thachuk and Y. Liu, eds., Cham, 2019, Springer International Publishing, pp. 140–158.

[6] B. BEHSAZ, J. MAŇUCH, AND L. STACHO, *Turing universality of step-wise and stage assembly at temperature 1*, in DNA Computing and Molecular Programming, D. Stefanovic and A. Turberfield, eds., Berlin, Heidelberg, 2012, Springer Berlin Heidelberg, pp. 1–11.

[7] F. BENZ AND T. KÖTZING, *An effective heuristic for the smallest grammar problem*, in Proceedings of the 15th annual conference on genetic and evolutionary computation, 2013, pp. 487–494.

[8] H. BUHRMAN, L. FORTNOW, AND S. LAPLANTE, *Resource-bounded kolmogorov complexity revisited*, SIAM Journal on Computing, 31 (2001), pp. 887–905.

[9] D. CABALLERO, T. GOMEZ, R. SCHWELLER, AND T. WYLIE, *Verification and Computation in Restricted Tile Automata*, in 26th International Conference on DNA Computing and Molecular Programming (DNA 26), C. Geary and M. J. Patitz, eds., vol. 174 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2020, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 10:1–10:18.

[10] S. CANNON, E. D. DEMAINE, M. L. DEMAINE, S. EISENSTAT, M. J. PATITZ, R. T. SCHWELLER, S. M. SUMMERS, AND A. WINSLOW, *Two Hands Are Better Than One (up to constant factors): Self-Assembly In The 2HAM vs. aTAM*, in 30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013), vol. 20 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, pp. 172–184.

[11] A. A. CANTU, A. LUCHSINGER, R. SCHWELLER, AND T. WYLIE, *Signal passing self-assembly simulates tile automata*, in 31st International Symposium on Algorithms and Computation (ISAAC 2020), Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[12] C. CHALK, A. LUCHSINGER, E. MARTINEZ, R. SCHWELLER, A. WINSLOW, AND T. WYLIE, *Freezing simulates non-freezing tile automata*, in DNA Computing and Molecular Programming, D. Doty and H. Dietz, eds., Cham, 2018, Springer International Publishing, pp. 155–172.

[13] C. CHALK, E. MARTINEZ, R. SCHWELLER, L. VEGA, A. WINSLOW, AND T. WYLIE, *Optimal staged self-assembly of general shapes*, Algorithmica, 80 (2018), pp. 1383–1409.

[14] G. CHATTERJEE, N. DALCHAU, R. A. MUSCAT, A. PHILLIPS, AND G. SEELIG, *A spatially localized architecture for fast and modular DNA computing*, Nature Nanotechnology, (2017).

[15] M. COOK, Y. FU, AND R. SCHWELLER, *Temperature 1 self-assembly: Deterministic assembly in 3d and probabilistic assembly in 2d*, in Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms, SIAM, 2011, pp. 570–589.

[16] E. D. DEMAINE, M. L. DEMAINE, S. P. FEKETE, M. ISHAQUE, E. RAFALIN, R. T. SCHWELLER, AND D. L. SOUVAINE, *Staged self-assembly: nanomanufacture of arbitrary shapes with o (1) glues*, Natural Computing, 7 (2008), pp. 347–370.

[17] E. D. DEMAINE, S. EISENSTAT, M. ISHAQUE, AND A. WINSLOW, *One-dimensional staged self-assembly*, in Proceedings of the 17th international conference on DNA computing and molecular programming, DNA'11, 2011, pp. 100–114.

[18] A. DENNUNZIO, E. FORMENTI, L. MANZONI, G. MAURI, AND A. E. PORRECA, *Computational complexity of finite asynchronous cellular automata*, Theoretical Computer Science, 664 (2017), pp. 131–143.

[19] D. DOTY, J. H. LUTZ, M. J. PATITZ, R. T. SCHWELLER, S. M. SUMMERS, AND D. WOODS, *The tile assembly model is intrinsically universal*, in 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science, IEEE, 2012, pp. 302–310.

[20] N. FATES, *A guided tour of asynchronous cellular automata*, in International Workshop on Cellular Automata and Discrete Complex Systems, Springer, 2013, pp. 15–30.

[21] B. FU, M. J. PATITZ, R. T. SCHWELLER, AND R. SHELINE, *Self-assembly with geometric tiles*, in International Colloquium on Automata, Languages, and Programming, Springer, 2012, pp. 714–725.

[22] D. FURCY, S. M. SUMMERS, AND L. WITHERS, *Improved Lower and Upper Bounds on the Tile Complexity of Uniquely Self-Assembling a Thin Rectangle Non-Cooperatively in 3D*, in 27th International Conference on DNA Computing and Molecular Programming (DNA 27), M. R. Lakin and P. Šulc, eds., vol. 205 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2021, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 4:1–4:18.

[23] O. GILBERT, J. HENDRICKS, M. J. PATITZ, AND T. A. ROGERS, *Computing in continuous space with self-assembling polygonal tiles*, in Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2016, pp. 937–956.

[24] E. GOLES, P.-E. MEUNIER, I. RAPAPORT, AND G. THEYSSIER, *Communication complexity and intrinsic universality in cellular automata*, Theoretical Computer Science, 412 (2011), pp. 2–21.

[25] E. GOLES, N. OLLINGER, AND G. THEYSSIER, *Introducing freezing cellular automata*, in Cellular Automata and Discrete Complex Systems, 21st International Workshop (AUTOMATA 2015), vol. 24, 2015, pp. 65–73.

[26] L. N. GREEN, H. K. SUBRAMANIAN, V. MARDANLOU, J. KIM, R. F. HARIADI, AND E. FRANCO, *Autonomous dynamic control of DNA nanostructure self-assembly*, Nature chemistry, 11 (2019), pp. 510–520.

[27] D. HADER AND M. J. PATITZ, *Geometric tiles and powers and limitations of geometric hindrance in self-assembly*, Natural Computing, 20 (2021), pp. 243–258.

[28] J. HENDRICKS, M. J. PATITZ, T. A. ROGERS, AND S. M. SUMMERS, *The power of duples (in self-assembly): It's not so hip to be square*, Theoretical Computer Science, 743 (2018), pp. 148–166.

[29] L. LONGPRÉ, *Resource bounded kolmogorov complexity, a link between computational complexity and information theory*, tech. rep., Cornell University, 1986.

[30] P.-E. MEUNIER, M. J. PATITZ, S. M. SUMMERS, G. THEYSSIER, A. WINSLOW, AND D. WOODS, *Intrinsic universality in tile self-assembly requires cooperation*, in Proceedings of the 2014 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2014, pp. 752–771.

[31] P.-E. MEUNIER AND D. REGNAULT, *Directed Non-Cooperative Tile Assembly Is Decidable*, in 27th International Conference on DNA Computing and Molecular Programming (DNA 27), M. R. Lakin and P. Šulc, eds., vol. 205 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2021, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 6:1–6:21.

[32] P.-E. MEUNIER AND D. WOODS, *The non-cooperative tile assembly model is not intrinsically universal or capable of bounded Turing machine simulation*, in Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, New York, NY, USA, 2017, Association for Computing Machinery, pp. 328–341.

[33] T. NEARY AND D. WOODS, *P-completeness of cellular automaton rule 110*, in Automata, Languages and Programming, M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, eds., Berlin, Heidelberg, 2006, Springer Berlin Heidelberg, pp. 132–143.

[34] N. OLLINGER AND G. THEYSSIER, *Freezing, bounded-change and convergent cellular automata*, arXiv preprint arXiv:1908.06751, (2019).

[35] J. E. PADILLA, M. J. PATITZ, R. T. SCHWELLER, N. C. SEEMAN, S. M. SUMMERS, AND X. ZHONG, *Asynchronous signal passing for tile self-assembly: Fuel efficient computation and efficient assembly of shapes*, International Journal of Foundations of Computer Science, 25 (2014), pp. 459–488.

[36] M. J. PATITZ, R. T. SCHWELLER, AND S. 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, Berlin, Heidelberg, 2011, Springer-Verlag, pp. 175–189.

[37] P. W. ROTHEMUND AND E. WINFREE, *The program-size complexity of self-assembled squares*, in Proceedings of the thirty-second annual ACM symposium on Theory of computing, 2000, pp. 459–468.

[38] N. SCHIEFER AND E. WINFREE, *Time complexity of computation and construction in the chemical reaction network-controlled tile assembly model*, in DNA Computing and Molecular Programming, Y. Rondelez and D. Woods, eds., Cham, 2016, Springer International Publishing, pp. 165–182.

[39] D. SOLOVEICHIK AND E. WINFREE, *Complexity of self-assembled shapes*, SIAM Journal on Computing, 36 (2007), pp. 1544–1569.

[40] A. J. THUBAGERE, W. LI, R. F. JOHNSON, Z. CHEN, S. DOROUDI, Y. L. LEE, G. IZATT, S. WITTMAN, N. SRINIVAS, D. WOODS, E. WINFREE, AND L. QIAN, *A cargo-sorting DNA robot*, Science, 357 (2017), p. eaan6558.

[41] E. WINFREE, *Algorithmic Self-Assembly of DNA*, PhD thesis, California Institute of Technology, June 1998.

[42] A. WINSLOW, *Staged self-assembly and polyomino context-free grammars*, Natural Computing, 14 (2015), pp. 293–302.

[43] D. WOODS, H.-L. CHEN, S. GOODFRIEND, N. DABBY, E. WINFREE, AND P. YIN, *Active self-assembly of algorithmic shapes and patterns in polylogarithmic time*, in Proceedings of the 4th conference on Innovations in Theoretical Computer Science, 2013, pp. 353–354.

[44] D. WOODS AND T. NEARY, *The complexity of small universal turing machines: A survey*, Theoretical Computer Science, 410 (2009), pp. 443–450.

[45] T. WORSCH, *Towards intrinsically universal asynchronous ca*, Natural Computing, 12 (2013), pp. 539–550.

BIOGRAPHICAL SKETCH

Sonya Cirlos was born and raised in the Rio Grande Valley. She has a background in molecular biology and worked at a biomedical research institute in Houston, TX before deciding to pursue her Masters in Computer Science. During her Masters studies, she co-founded a coding community organization, Frontera Devs, where her and other women in computer science host technical workshops and lead mentorship programs for undergraduate and high school students across the Rio Grande Valley. She currently works remotely for a telemedicine start up in Nigeria and plans to continue to use her programming and design skills to advance biomedical research and healthcare. Sonya can be contacted at sccirlos@gmail.com.