Contrasting Geometric Variations of Mathematical Models of Self-assembling Systems

Michael Sharp

University of Arkansas, Fayetteville

Follow this and additional works at: https://scholarworks.uark.edu/etd

Part of the Numerical Analysis and Scientific Computing Commons, Systems Architecture Commons, and the Theory and Algorithms Commons

Citation

This Thesis is brought to you for free and open access by ScholarWorks@UARK. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of ScholarWorks@UARK. For more information, please contact ccmiddle@uark.edu.
Contrasting Geometric Variations of Mathematical Models of Self-assembling Systems

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

by

Michael Sharp
University of Arkansas
Bachelor of Science in Computer Science, 2017

December 2019
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

Matthew Patitz, Ph.D.
Thesis Director

John Gauch, Ph.D.
Committee member

Khoa Luu, Ph.D.
Committee member
ABSTRACT

Self-assembly is the process by which complex systems are formed and behave due to the interactions of relatively simple units. In this thesis, we explore multiple augmentations of well known models of self-assembly to gain a better understanding of the roles that geometry and space play in their dynamics. We begin in the abstract Tile Assembly Model (aTAM) with some examples and a brief survey of previous results to provide a foundation. We then introduce the Geometric Thermodynamic Binding Network model, a model that focuses on the thermodynamic stability of its systems while utilizing geometrically rigid components (dissimilar to other thermodynamic models). We show that this model is computationally universal, an ability conjectured to be impossible in similar models with non-rigid components. We continue by introducing the Flexible Tile Assembly Model, a generalization of the 2D aTAM that allows bonds between tiles to flex and assemblies to therefore reconfigure. We show how systems in this model can deterministically assemble shapes that adhere to a number of certain restrictions. Finally, we introduce the Spatial abstract Tile Assembly Model, a variation of the 3D aTAM that restricts tiles from attaching without a diffusion path. We show that this model is intrinsically universal, a property of computational models to simulate themselves which has been shown for the 3D aTAM and other similar models. We conclude this thesis with a summary of the presented results, a brief impact analysis, and potential directions for future research within this area.
ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Matthew Patitz for giving me the opportunity to go to graduate school and supporting me throughout. Matt has been an wonderful mentor and taught me an immense amount, both in the lab and out. He is an incredible achiever in all aspects of his life and has easily the strongest work ethic I’ve ever witnessed. I’m grateful for all the effort Matt has put into seeing me succeed as one of his students for the whole of my college career, as well as being a great friend throughout it all.

Additionally, I would like to thank the exceptional group of individuals that comprise the self-assembly lab at the University of Arkansas. This group includes Daniel Hader, Trent Rogers, Jacob Hendricks and all the other students that I have come to know through Matt’s research group. These bright people made the lab conducive to great research while also contributing to a fun work environment. I would also like to thank all the members of the broader self-assembly community as well for bringing me in and allowing me to experience this exciting and unique field of work through collaboration with them.

And finally, I would like to thank all the friends I have made throughout my time in Fayetteville for making it the best six (and a half, whoops) years I could have imagined. Hotz Hall, 1005 Hill Place, Cardinal 431, and 1173 Maxwell were all just as important places to me as a student as Bell Engineering or JBHT were. These friends pushed me to grow in all areas of my life and made me into who I am. The engravings of our names together on senior walk will always commemorate the experiences I’ve gotten to share with them.
DEDICATION

I would like to dedicate this thesis to my parents, Max and Donna Sharp. In the wake of completing my degree and all the effort I’ve put into it, it’s easy for me to forget the massive amounts of work and sacrifice that you both have put into raising me and Shelby and setting us up to accomplish our goals and succeed in life. I’m unbelievably lucky to have you as my parents. Thank you for all that you have done for me. I love you guys.
# TABLE OF CONTENTS

1 Introduction ................................................................. 1

2 Previous Models and Results in Self-assembly .......................... 6
  2.1 Basic example system .................................................. 6
  2.2 Formal definition of the abstract Tile Assembly Model .......... 8
  2.3 Binary counter example ................................................ 10
  2.4 Important prior results in the aTAM and aTAM variations ........ 12

3 Spatial Computation without Kinetic Pathways ....................... 16
  3.1 Formal definition of the Geometric Thermodynamic Binding Network .. 18
  3.2 Efficient simulation of Turing machines in GTBN’s .................. 20
    3.2.1 Construction components ........................................ 24
    3.2.2 Desired configuration ............................................ 32
    3.2.3 Incorrect computations ........................................... 35
    3.2.4 Seedless polymers ............................................... 41

4 Utilizing 2D Components to Build 3D Structures ...................... 46
  4.1 Formal definition of the Flexible Tile Assembly Model .......... 46
  4.2 Controlling flexibility to build structures ....................... 53

5 The Effects of Modeling Diffusion on Simulation ..................... 66
  5.1 Formal definition of the Spatial abstract Tile Assembly Model .... 67
    5.1.1 Diffusion constraint ............................................. 67
    5.1.2 Simulation ....................................................... 68
    5.1.3 Intrinsic universality .......................................... 70
  5.2 Intrinsic universality in the Spatial abstract Tile Assembly Model ... 71
    5.2.1 Our augmented construction ...................................... 76
    5.2.2 Incoming genome and external communication .................. 78
    5.2.3 Outgoing genome and external communication .................. 79
    5.2.4 Receiver and piping ............................................. 80
    5.2.5 Growing the side faces ......................................... 81
    5.2.6 Differentiation and activation ................................... 83
    5.2.7 Seed and representation functions ............................... 86
    5.2.8 Correctness ...................................................... 87

6 Conclusion ................................................................. 91
  6.1 Summary of results and their impacts ................................ 91
  6.2 Open problems and future work ..................................... 92

Bibliography ............................................................... 94
LIST OF FIGURES

Figure 2.1: An example tile set. Here we have four tile types. We assume the system has an infinite number of each type. In this example, the number of bumps on a tile’s edge (one or two) indicates the strength of that edge’s corresponding glue, and the text closest to the edge indicates the label of that glue. ........................................... 7

Figure 2.2: An example system. (1) the seed. (2 and 3) intermediate assemblies. (4) the terminal assembly. ................................................................. 7

Figure 2.3: The tile set for the binary counter system. These tile types, a designation of the grey “S” tile as the seed, and the setting of the temperature to two make up the system definition. ........................................ 11

Figure 2.4: A producible assembly in the binary counter system. Each grey “B” and “R” tile attaches independently to the grey “S” seed tile through the blue double strength glues and then repeatedly to each other. Each “0” and “1” tile attaches through the cooperation of two single strength glues. The pattern shown here can repeat til infinity. .......................... 12

Figure 3.1: This graph shows gaps in both enthalpy and entropy of different polymers in our system. The most favorable polymer should always be the paired computations.............................................. 23

Figure 3.2: On the top is the illustration of the seed we use in other figures. The bottom diagram shows the specific domains that are present on the seed supertile. These include two tape extension domains, two padding domains, the TM input encoding, and the final pairing domains. .......... 24

Figure 3.3: The addition of all correct pieces to the seed are enthalpy and entropy neutral. Each breaks two bonds with the cap and forms two more with the computation while also remaining at two polymers before and after the transition. .......................................................... 32

Figure 3.4: On the left is a completed computation that has not yet been paired. Compared to the fully capped state, it has the same number of bonds and polymers, meaning it still has neutral enthalpy and entropy. ............. 36

Figure 3.5: This illustrates the pairing of two complete computations. The dislodging of the seed caps creates an entropy bonus of one. This bonus is what makes the final complete paired construction favorable. ......................... 37
Figure 3.6: On the left is an example of a spurious computation with exposed domains. Because these domains are unbounded, there is an enthalpy gap between this polymer with its detached caps and a fully capped state. On the right is a special case where all the exposed input domains are still bound to their caps. In this case, there is an entropy gap with a fully capped state. To clarify, these two subsystems are not being compared to each other, but the statistics below each image are the comparisons between each polymer and a fully capped state.

Figure 4.1: Possible normal vectors of pairs of tiles. Those in (a) are compatible and allow a bond to form between complementary glues in the orientations “Up”, “Down”, and “Straight”, respectively. Those in (b) are not compatible.

Figure 4.2: Here we see an assembly, a valid configuration, and an invalid configuration. In the third image, because of the orientations of bonds 1, 2, and 4, bond 3 is between two tiles that are not connected, making the configuration invalid.

Figure 4.3: All possible polycubes that can fit inside of a $2 \times 2 \times 2$ space, and furthermore, all possible vertex types that could exist on a polycube.

Figure 4.4: Illustration of (a) an original edge, (b) a convex vertex, (c) a concave vertex, from one unique perspective, and (d) a concave vertex, from another perspective.

Figure 4.5: The loops of tiles that form each complex vertex.

Figure 4.6: An assembling edge frame starting from a potential seed. Each edge grows up to a vertex and into other edges until the whole frame has filled out.

Figure 4.7: Example of a shape with multiple edge frames. The left shows the shape assembled with tiles. The right shows the outline of the shape and how it is broken into 3 disconnected pieces.

Figure 5.1: Illustration of the steps in which a macrotile goes from mapping to empty space to differentiating into representing a specific tile type in the simulated system and propagating that information to its neighbors.

Figure 5.2: Illustration of the growth process of the genome from inside one macrotile location into neighboring macrotile locations.

Figure 5.3: The pipe intersection. This module is important in the simulation of the spatial constraint because it allows multiple paths to be cut off simultaneously when the macrotile differentiates.
Figure 5.4: The interception gadget works by growing out from an incoming genome path into the path of the future pipe that will extend in the same direction. Once the pipe grows, the original sequence of tile placements is blocked, and it instead cooperates with the interception gadget to continue growing but with a hole in the side toward the original genome path. Then, whenever the macrotile differentiates and tiles start growing through the piping, a signal can grow out of the hole and along the genome to activate its (and the corresponding external communication datapath’s) propagation into the neighboring macrotile in the “up” direction. The path of this signal is shown in yellow.

Figure 5.5: For this example, we make it so that the scale factor only needs to be encoded into the bottom face so that it knows how far to grow outwards to get to each side face. Each pipe that grows outwards in the N,E,S,W directions is tethered to the bottom face so that it also grows exactly up to each side face. Each black tile in the corner then grows up a pole (as seen on the left in Figure 5.6) that begins the growth of the bottom strip of each side face.

Figure 5.6: The bottom strip of each side boundary. Growth starts from the special tiles on the left. Generic filler tiles grow either (a) around the incoming genome and external communication or (b) over the slots to block the incoming datapaths from coming in later. The special tile in the middle of the bottom connects to the end of the piping (thereby preventing the filler tiles from growing over it). A constant number of hardcoded tiles then count over to the slots designated for the outgoing genome and external communication. The filler tiles must then wait for these datapaths to come in before they can continue on. The tile in the upper-rightmost corner is guaranteed to be the last placed.

Figure 5.7: Once the bottom strip has of each side face has completely filled out, one of the corner tiles starts the growth of the rest of the side faces by growing a one tile wide path around the top of each bottom strip. Each bottom strip can be certain to have completely by the time one loop has been made. Then, the path steps one in the “up” direction (through cooperation with the first tile in the path) and starts another path. This continue until the path reaches the receiver in the upper right corner of the side face on which the first path started. Once this receiver is hit, all the blocking protocol is certain to have completed, and the macrotile is clear to differentiate.

Figure 5.8: A diagram depicting the timing dependencies present in our implementation. Once the bottom strips are tiled, all the timing dependencies are rectified and the rest of the process is sequential.
1 Introduction

Self-assembly is the process by which relatively simple components organize through local interaction to exhibit complex behaviors and build complex structures. Examples in nature include the symmetric formation of snowflakes and the organization of proteins into viruses. It has spurred a rapidly growing and developing field of research consisting of both abstracted models and physical implementations. On the theoretical side, researchers have utilized mathematics and computational theory to develop models that explore the inherent capabilities and limitations of these systems [1–5]. These models abstract away the physical substrate in order to focus on certain aspects and abilities of the systems ranging from shape building [6] to computational simulations [7]. Meanwhile, experimental researchers have run through an assortment of methods for synthesizing their own systems in efforts to make them more useful, powerful, and consistent [8–12]. These implementations have utilized a variety of substrates including specific chemicals and nucleic acids and have yielded systems with a mixture of abilities ranging from simulating integrated circuits [13] to building reconfigurable nanostructures [14]. These opposing sides of research within the field of self-assembly continually influence each other, with the theoretical work guiding the experimental design and the experimental work inspiring the theoretical modeling.

In this thesis, we will look at a group of models that we have introduced and developed to study a specific set of properties of self-assembling systems. These models are all variations of already established models that differ in some geometric aspect, whether it be through the system components themselves or the dynamics of how the systems assemble structures.
The goal of this work is to learn how these model differences affect the abilities of these systems to (a) compute, (b) build structures, and (c) simulate other systems.

We will begin in the following chapter with a short introduction to the abstract Tile Assembly Model (aTAM). This model spurred a subcategory of general self-assembly models called tile assembly models and is the most developed of any model with this designation. The latter two of the three models that we will introduce also fall into this subcategory. Being the most widely studied, the aTAM also has the largest plethora of results, leading us to have a relatively clear understanding of what it can and cannot do. This makes it an ideal foundation with which to explore other models, given that the techniques and constructions used to establish these results can be adapted to these new model variations in an attempt to develop analogous results.

Next, we study the Geometric Thermodynamic Binding Network (GTBN) model. This model is actually an adaptation of the general Thermodynamic Binding Network (TBN) model with the addition of geometric constraints on the system components, thereby making them more similar to aTAM components. However, both the TBN model and the GTBN model differ from the aTAM in that they focus on the relative thermodynamic stability of different configurations a given system can exist in, rather than the producible assemblies that a system can yield after a series of bindings between components. The goal of introducing and studying the GTBN model is to address the discrepancies in the computational powers between the aTAM and TBN model. Specifically, it has been shown that the aTAM is capable of universal computation through the simulation of other computational models, while it is conjectured that the TBN model is unable to do the same (especially through the same techniques). By adding the geometric constraint, we show that the the GTBN model is
also able to perform these computations using systems that are analogous to computational aTAM systems. This result gives us evidence that the current inability to compute in the TBN model stems from the geometric differences between it and the aTAM as opposed to any other differences, such as the contrasting dynamics that were previously mentioned. In addition, it gives us a notion that, when a geometric constraint is included, slightly augmented aTAM techniques for computation produce assemblies that are considered stable even when viewed from a thermodynamic perspective, meaning these systems should also be more thermodynamically favorable when physically implemented within a test tube. The introduction of the GTBN model, the result we show in Chapter 3, and other results were published in the proceedings of the 17th International Conference on Unconventional Computation and Natural Computation (UCNC 2018) [15].

Then, we look at the Flexible Tile Assembly Model (FTAM). This model is a generalization of the 2D aTAM that allows the bonds between components to “flex” and create 3D structures. We show that the model is capable of deterministically building a set of 3D structures that meet a number of specific criteria. One motivation behind this model is that it provides a foundation for us to study problems such as protein structure prediction from the perspective of tile assembly models. Protein structure prediction is the computational task of determining the final structure of a protein given a sequence of amino acids. It has been shown that this problem is computationally intractable. However, the hope is that the FTAM can give us the tools and language to further study protein structure prediction and other similar problems. Additionally, the FTAM is motivated by experimental research showing that the implementation of flexible bonds is physically realizable [14]. Therefore, having a model that incorporates this feature into its model definition allows us to study
the impacts that a tangible implementation of the feature would have on the capabilities and limitations of the physical systems. The introduction of the FTAM, the result we show in Chapter 4, and other results were published in the proceedings of the 24th International Conference on DNA Computing and Molecular Programming (DNA 24) [16].

Finally, we explore the Spatial abstract Tile Assembly Model (Spatial aTAM). This model is a variation of the 3D aTAM with the addition of a diffusion constraint, i.e. component are prevented from binding within enclosed cavities of a structure. The motivation for this model is to see if the results in the 3D aTAM still hold whenever this realistic constraint is taken into account. In other words, if the results don’t hold, they probably aren’t realizable in tangible systems, whereas if the results do hold, that is evidence that they could potentially be implemented and at least won’t be prevented due to diffusion issues. The result we look at in this thesis is the property of a model to be intrinsically universal (IU). Within an IU model, there exists a single system that is capable of simulating any other system within the model. For example, Turing machines are IU, and we show in [17] that the 3D aTAM is IU. Additionally, studying this property in the Spatial aTAM helps us to compare the relative computational power between self-assembly models and computational models in general. The introduction of the Spatial aTAM, the result we show in Chapter 5, and other results were accepted for (and are to be published in) the proceedings of the ACM-SIAM Symposium on Discrete Algorithms 2020 (SODA 2020) [17].

In the big picture, the research in this thesis will give us better resolution in our understanding of the possibilities within self-assembling systems, both theoretically and experimentally. The impact of this work is that other researchers will be able to use our models and results to assist in both developing their own abstract models and designing their own
experiments. Our deliverables should also help in areas such as the thermodynamic analysis of self-assembling systems, the predictive study of folding processes, and the power comparison of varying computational models. Overall, the hope is that our work will one day lead to the realization of self-assembling systems with previously unmatched complexity and capability, thereby redefining our notion of what is possible on the nanoscale.
2 Previous Models and Results in Self-assembly

As previously mentioned, the foundational model of this thesis, i.e. the model that gives us some context within which to analyze the other three models, will be the abstract Tile Assembly Model (aTAM). This model was introduced by Erik Winfree in his Ph.D. dissertation at the California Institute of Technology [1]. We will begin with a basic example of an aTAM system to gain an intuition of the model and its dynamics. We will then give formal definitions, continue into a more advanced example, and then go over important results in the model and its variations.

2.1 Basic example system

Before we get into formal definitions for the model, we believe it would be beneficial to readers unfamiliar with the aTAM to first walk through a basic example. There are three pieces to the definitions of an aTAM system: the tile set, the seed, and the temperature. The tile set is the finite set of available tile types (an example is shown in Figure 2.1). Each tile type is a square with a “glue” on each edge. An infinite number of each tile type (called “tiles”) is present in the system. The system begins with the seed (shown in Figure 2.2(1)) and additional tiles can attach one at a time if their glues can match with enough “strength” (as defined by the “temperature”, or binding threshold, of the system).

In this example, the tile set is shown in Figure 2.1, the seed is the yellow tile, and the temperature is two. Figure 2.2 depicts the assembly process from start to finish. From the seed (1), the assembly can grow by attaching the red and blue tiles utilizing double strength
Figure 2.1: An example tile set. Here we have four tile types. We assume the system has an infinite number of each type. In this example, the number of bumps on a tile’s edge (one or two) indicates the strength of that edge’s corresponding glue, and the text closest to the edge indicates the label of that glue.

Figure 2.2: An example system. (1) the seed. (2 and 3) intermediate assemblies. (4) the terminal assembly.

gleues (2 and 3). Since these glues have strength two and the system temperature is two, they can bind independently without assistance from other glues or tiles. Either the red or blue can non-deterministically attach first, but the only possible attachment after one of these tiles is the other tile, meaning the assembly will end up in the state shown in (3) regardless of which of these two tiles attaches first. Once both have attached, the green tile can then “cooperate” with both via single strength glues to attach itself (4). Since there are no more exposed glues, no more tiles can attach and the assembly is terminal.
2.2 Formal definition of the abstract Tile Assembly Model

We will now formally define the aTAM. Since we will later introduce variations of both the 2D and the 3D aTAM, we will provide dimension agnostic definitions here.

Fix an alphabet $\Sigma$. $\Sigma^*$ is the set of finite strings over $\Sigma$. $\mathbb{Z}$, $\mathbb{Z}^+$, and $\mathbb{N}$ denote the set of integers, positive integers, and nonnegative integers, respectively. Let $d \in \{2, 3\}$. Given $V \subseteq \mathbb{Z}^d$, the full grid graph of $V$ is the undirected graph $G_V^f = (V, E)$, and for all $x = (x_0, \ldots, x_{d-1}), y = (y_0, \ldots, y_{d-1}) \in V$, $\{x, y\} \in E \iff \|x - y\| = 1$; i.e., if and only if $x$ and $y$ are adjacent on the $d$-dimensional integer Cartesian space.

A $d$-dimensional tile type is a tuple $t \in (\Sigma^* \times \mathbb{N})^{2d}$; e.g., a unit square (or cube) with four (or six) sides listed in some standardized order, each side having a glue $g \in \Sigma^* \times \mathbb{N}$ consisting of a finite string label and non-negative integer strength. We assume a finite set of tile types, but an infinite number of copies of each tile type, each copy referred to as a tile (either a 2D square or 3D cube tile type). A $d$-dimensional tile set is a set of $d$-dimensional tile types and is written as $d$-$T$. A tile set $T$ is a set of $d$-dimensional tile types for some $d \in \{2, 3\}$.

A $d$-configuration is a (possibly empty) arrangement of tiles on the integer lattice $\mathbb{Z}^d$, i.e., a partial function $\alpha : \mathbb{Z}^d \rightarrow T$. A configuration $\alpha$ is a $d$-configuration for some $d \in \{2, 3\}$. Two adjacent tiles in a configuration interact, or are attached, if the glues on their abutting sides are equal (in both label and strength) and have positive strength. Each configuration $\alpha$ induces a binding graph $G^b_\alpha$, a grid graph whose vertices are positions occupied by tiles, according to $\alpha$, with an edge between two vertices if the tiles at those vertices interact. A $d$-assembly is a connected non-empty configuration, i.e., a partial function
\( \alpha : \mathbb{Z}^d \rightarrow T \) such that \( G^\text{dom}_\alpha \) is connected and \( \text{dom} \alpha \neq \emptyset \). An assembly is a \( d \)-assembly for some \( d \in \{2, 3\} \). The shape \( S_\alpha \subseteq \mathbb{Z}^d \) of \( \alpha \) is \( \text{dom}_\alpha \).

Given \( \tau \in \mathbb{Z}^+ \), \( \alpha \) is \( \tau \)-stable if every cut of \( G^b_\alpha \) has weight at least \( \tau \), where the weight of an edge is the strength of the glue it represents. When \( \tau \) is clear from context, we say \( \alpha \) is stable. Given two assemblies \( \alpha, \beta \), we say \( \alpha \) is a subassembly of \( \beta \), and we write \( \alpha \sqsubseteq \beta \), if \( S_\alpha \subseteq S_\beta \) and, for all points \( p \in S_\alpha \), \( \alpha(p) = \beta(p) \).

A \( d \)-dimensional tile assembly system (\( d \)-TAS) is a triple \( d \cdot T = (d \cdot T, \sigma, \tau) \), where \( d \cdot T \) is a finite set of \( d \)-dimensional tile types, \( \sigma : \mathbb{Z}^d \rightarrow T \) is the finite, \( \tau \)-stable, \( d \)-dimensional seed assembly, and \( \tau \in \mathbb{Z}^+ \) is the temperature. The triple \( T = (T, \sigma, \tau) \) is a TAS if it is a \( d \)-TAS for some \( d \in \{2, 3\} \). Given two \( \tau \)-stable assemblies \( \alpha, \beta \), we write \( \alpha \rightarrow^T T \beta \) if \( \alpha \sqsubseteq \beta \) and \( |S_\beta \setminus S_\alpha| = 1 \). In this case we say \( \alpha \) \( \mathcal{T} \)-produces \( \beta \) in one step. If \( \alpha \rightarrow^\mathcal{T} T \beta \), \( S_\beta \setminus S_\alpha = \{p\} \), and \( t = \beta(p) \), we write \( \beta = \alpha + (p \mapsto t) \). The \( \mathcal{T} \)-frontier of \( \alpha \) is the set \( \partial \mathcal{T} \alpha = \bigcup_{\alpha \rightarrow^\mathcal{T} T \beta} S_\beta \setminus S_\alpha \), the set of empty locations at which a tile could stably attach to \( \alpha \). The \( t \)-frontier \( \partial_t \alpha \subseteq \partial \alpha \) of \( \alpha \) is the set \( \{ p \in \partial \alpha \mid \alpha \rightarrow^\mathcal{T} T \beta \text{ and } \beta(p) = t \} \).

Let \( \mathcal{A}^T \) denote the set of all assemblies of tiles from \( T \), and let \( \mathcal{A}^T_{<\infty} \) denote the set of finite assemblies of tiles from \( T \). A sequence of \( k \in \mathbb{Z}^+ \cup \{\infty\} \) assemblies \( \alpha_0, \alpha_1, \ldots \) over \( \mathcal{A}^T \) is a \( \mathcal{T} \)-assembly sequence if, for all \( 1 \leq i < k \), \( \alpha_{i-1} \rightarrow^\mathcal{T} T \alpha_i \). The result of an assembly sequence is the unique limiting assembly (for a finite sequence, this is the final assembly in the sequence).

For the purposes of notational convenience, we equivalently define an assembly sequence \( \alpha \) as an initial assembly \( \alpha_0 \), followed by a (possibly infinite) sequence of \( k \in \mathbb{Z}^+ \cup \{\infty\} \) single-tile-placements, or pairs of grid points and tile types \( (x_0, t_0), (x_1, t_1), \ldots \) satisfying, for each \( 0 \leq i < k \), \( \alpha_{i+1} = \alpha_i + (x_i \mapsto t_i) \), and for an assembly sequence \( A \) will use \( A[i] \) to
specify the \( i \)th pair, \((x_i, t_i)\). When the initial assembly \( \alpha_0 \) is clear from context, an assembly sequence \( \alpha \) can be stated strictly in terms of a sequence of single-tile-placements.

We write \( \alpha \rightarrow^T \beta \), and we say \( \alpha \) \( T \)-produces \( \beta \) (in 0 or more steps) if there is a \( T \)-assembly sequence \( \alpha_0, \alpha_1, \ldots \) of length \( k = |S_\beta \setminus S_\alpha| + 1 \) such that (1) \( \alpha = \alpha_0 \), (2) \( S_\beta = \bigcup_{0 \leq i < k} S_{\alpha_i} \), and (3) for all \( 0 \leq i < k \), \( \alpha_i \subseteq \beta \). If \( k \) is finite then it is routine to verify that \( \beta = \alpha_{k-1} \). We say \( \alpha \) is \( T \)-producible if \( \sigma \rightarrow^T \alpha \), and we write \( \mathcal{A}[T] \) to denote the set of \( T \)-producible assemblies. The relation \( \rightarrow^T \) is a partial order on \( \mathcal{A}[T] \).

Given a TAS \( T = (T, \sigma, \tau) \), an assembly \( \alpha \) is \( T \)-producible, or producible if \( T \) is clear from context, if either \( \alpha = \sigma \), or \( \alpha \) results from the \( \tau \)-stable attachment of a single tile to a producible assembly ("\( \tau \)-stable attachment" meaning that the cut separating the tile from the rest of the assembly has strength \( \geq \tau \), which implies by induction that \( \alpha \) is \( \tau \)-stable). An assembly \( \alpha \) is \( T \)-terminal if \( \alpha \) is \( \tau \)-stable and \( \partial^T \alpha = \emptyset \). We write \( \mathcal{A}_{\square}[T] \subseteq \mathcal{A}[T] \) to denote the set of \( T \)-producible, \( T \)-terminal assemblies. If \( |\mathcal{A}_{\square}[T]| = 1 \) then \( T \) is said to be directed.

When \( T \) is clear from context, we may omit \( T \) from the notation above and instead write \( \rightarrow_1, \rightarrow, \partial \alpha \), assembly sequence, produces, producible, and terminal.

### 2.3 Binary counter example

We’ll now give a a more advanced example of an aTAM system in an effort to illustrate the power inherent in the aTAM model. This system will be temperature two and will begin with a single-tile seed. The tile set is shown in Figure 2.3. The strength of each glue is determined by the color (white for single strength, blue for double strength) and the text closest to the edge indicates the label of that glue. The assembly process begins from a
Figure 2.3: The tile set for the binary counter system. These tile types, a designation of the grey “S” tile as the seed, and the setting of the temperature to two make up the system definition.

seed, designated in the picture with the “S” label. From the seed, the grey tiles of type “B” and “R” can attach indefinitely in the west and north directions, respectively. Next, white and red tiles of type “0” and “1” can cooperate with the “B” and “R” tiles and each other to attach with strength two to the already formed assembly. After a number of these attachments, a possible assembly that could be produced is shown in Figure 2.4.

If you look at the glue labels in the white and red tiles on the bottom in Figure 2.3, you’ll notice that the glue labels actually implement the logic of a half adder. In this analogy, the east and south glues are the inputs to the adder, while the north and west glues are the sum and carry bits, respectively. Because of this design, the attachment of these tiles will actually “compute” a binary counter. If you look at each row independently in Figure 2.4, you’ll notice that the tile labels encode a binary number. The encoded number of each row is one larger than the row immediately below it, with the bottom row encoding zero.

Note that this system produces an infinite assembly since there are an infinite number of each tile type and there will always be locations on the assembly in which these tiles can attach, i.e. there will always be a non-empty frontier. This system is also directed because
Figure 2.4: A producible assembly in the binary counter system. Each grey “B” and “R” tile attaches independently to the grey “S” seed tile through the blue double strength glues and then repeatedly to each other. Each “0” and “1” tile attaches through the cooperation of two single strength glues. The pattern shown here can repeat til infinity.

there is only ever one tile type that can attach in any given location on the assembly. This means that, albeit infinite, there is only one terminal assembly that can ever be produced by this system.

2.4 Important prior results in the aTAM and aTAM variations

Now, we will go over some important prior results in the aTAM.

Erik Winfree also showed in [1] that the aTAM is computationally universal. This is done through simulation of computationally universal cellular automata systems. Additionally, it has been shown how to translate Turing machine definitions into computational aTAM systems (e.g. [18,19]). In these simulations, tiles within an aTAM system can be de-
signed to simulate state changes of a Turing machine and its tape. The seed can then encode the input to the Turing machine and allow the individual tiles to perform the computations by systematically attaching.

These demonstrations of computational ability in the aTAM, however, are all in cooperative environments, i.e. temperature two. There have been a slew of papers addressing the widely conjectured claim that non-cooperative 2D aTAM systems, i.e. temperature one, are incapable of this universal computation [20, 21]. This is interesting because a number of variations on the non-cooperative 2D aTAM gain this power back through their augmented model definitions, similar to the research we do in this thesis. We’ll talk here about some of these models that are geometric in how they differ from the traditional 2D aTAM.

The first example is the Polygonal Tile Assembly Model [22] which uses tiles of different polygons as opposed to solely square tiles. In the Polygonal TAM, it was shown that systems that solely use tiles of a regular polygonal shape with any number of sides greater than 6 are capable of universal computation even in non-cooperative environments. They also show a number of classes of systems that use tiles which are irregular polygons that are also capable of non-cooperative universal computation.

Similar to the last model, the Polyomino Tile Assembly Model [23] also uses non-square shaped tiles. However, instead of polygons, the Polyomino TAM uses polyominoes, i.e. shapes created from connecting unit squares such that their centers are all 1 unit from their neighbors (their edges line up). It was shown that systems in the Polyomino TAM, as long as they utilize polyomino tiles of a size 3 or larger, are able to compute, as well as systems that utilize at least two distinct polyomino shapes in their tile sets.

Going back to traditional tiles that are shaped as unit squares, the restricted glue Tile
Assembly Model [24] is a variation of the 2D aTAM that includes a single general negative strength glue. It was shown that this is also enough to allow for non-cooperative universal computation, as well as uniquely producing $N \times N$ squares with $O(\log N)$ tile types.

Finally, even though non-cooperative 2D assembly in the aTAM is conjectured to not be computationally universal, [25] showed that, it only requires one additional plane in the third dimension for the aTAM to perform universal computation. The paper [25] also showed how non-cooperative 2D aTAM assembly can perform “probabilistic” computations.

Moving on from computational universality, shape building is another topic of research in the aTAM, the efficiency of which is usually measured in terms of tile type complexity, i.e. the number of tile types in the tile set. The paper [26] shows a tight bound for the self-assembly of $N \times N$ squares, which is $O(\log(N)/\log(\log(N))$ tile types. It is conjectured that non-cooperative aTAM systems require $O(n)$ tile types. The paper [27] gives a method to design an aTAM system that can build a scaled-up version of any specific shape that utilizes a tile set with size on the order of the Kolmogorov complexity of the shape. The Kolmogorov complexity of a shape is the theoretical smallest program that can output the shape’s definition, i.e. the points of the shape.

Limitations of shape building in the aTAM have also been explored. The paper [28] proves that the discrete Sierpinski triangle cannot be strictly self-assembled in the aTAM. Strict self-assembly means that only locations in the shape can be tiled and nowhere else, as opposed to having a subset of labelled tiles in the shape. Self-similar fractals are more generally explored in [29].

Research is also done on more general computational complexity in the aTAM. The paper [30] explores the power of nondeterminism in the aTAM. The paper [31] shows that
designing a tile set that assembles a binary pattern, otherwise known as \textit{2-PATS}, is NP-hard.

In [7], it was shown that the aTAM is intrinisically universal. This means that there exists a single aTAM tile set that can be programmed through specific seeds to simulate any other arbitrary aTAM system. This is done by encoding the definition of the system to be simulated into the seed of the simulating system and allowing general tile types to compute the simulated dynamics and create a representative assembly modulo some scale factor. This notion, as well as the idea of one system “simulating” another, will be formally defined later.

However, similar to the claim that the non-cooperative 2D aTAM is not computationally universal, it is also shown that the non-cooperative aTAM (2D and 3D) cannot simulate the cooperative aTAM [32]. This formally shows how the jump from non-cooperative to cooperative environments “improves the range of dynamics and behaviors” that can exist in these tile assembly systems.

Additionally, further research of intrinsic universality in the cooperative 2D aTAM has shown that the directed aTAM, the subset of aTAM systems that produce only one terminal assembly each, is not intrinsically universal [33]. This means there is no single tile set capable of universal simulation of directed aTAM systems while itself remaining directed (the construction presented in [7] had inherent non-determinism).

For a more comprehensive overview of the aTAM and results within the model, we refer the reader to the following papers [34–36].
3 Spatial Computation without Kinetic Pathways

Our first model we contrast with the abstract Tile Assembly Model is the Thermodynamic Binding Network (TBN) model [5] [37]. As opposed to the aTAM which is a kinetic model, the TBN model is an thermodynamic model. This means that, rather than trying to predict what will happen in an implemented system through a series of assemblies, the TBN model tries to predict the most favorable state that the system can exist in, irrespective of any process for the units within the systems to organize into that state. In other words, the aTAM is more useful in studying kinetic pathways from small seed assemblies to larger terminal assemblies with the addition of tiles one-at-a-time, while the TBN model is more useful in studying the relative thermodynamic stability of different configurations a system can exist in.

As opposed to systems composed of tiles and assemblies, TBN systems are composed of monomers and polymers. Whereas tiles have a rigid shapes (squares, cubes, etc.) and locations within a tile assembly system, a monomer is instead defined to be just a multiset of domains and codomains (similar to glues) with no notion of geometry or position. Therefore, a monomer with an unbound domain can bind to any other monomer with a corresponding unbound codomain (a codomain is a domain which is complementary to a given domain). A set of connected monomers is called a polymer. Whereas aTAM systems begin with a seed to which tiles can attach one-at-a-time through a series of assembly steps, TBN systems don’t have a notion of “assembly steps” or an “assembly process”. Instead, TBN systems have a finite number of each monomer and can be analyzed through the set of all possible
configurations, i.e. all valid matchings between complementary domains and codomains in the system. The motivation of this analysis is to find the most thermodynamically stable configuration that a system can exist in. So far, this state has been modeled by the configuration that (a) maximized the number of bonds formed (enthalpy) and (b) the number of polymers formed (entropy). Additionally, to relate these two measurements in the computation of a configuration’s overall stability, previous studies have generally considered the case where enthalpy is infinitely favored over entropy. This means that a configuration with more bonds will always be more favored over a configuration with less bonds, regardless of the relative number of polymers between the two, i.e. entropy is only a tiebreaker for whenever multiple configurations have the same enthalpy.

As stated in Section 2.4, the aTAM has been proven to be a computationally universal model, i.e. given some representation function, for any valid Turing machine, there exists an aTAM system that will simulate the logic of that Turing machine through the dynamics of its assembly process. While not proven impossible, it has been shown that the traditional TBN model cannot simulate arbitrary Turing machines using a naive conversion of an aTAM system [15]. However, given just this information, it is unclear if the discrepancy in computational power stems from the lack of an assembly process or from the lack of geometry and position of monomers. Therefore, to address this question, we introduce a new variation of the TBN model called the Geometric Thermodynamic Binding Network (GTBN) model. This new model will give us additional tools to work with in arguing about the ability of TBN model variations to compute. In this chapter, we will define the model, and then we will build a GTBN construction similar to computation-designed aTAM systems.
3.1 Formal definition of the Geometric Thermodynamic Binding Network

Here we give definitions for the GTBN model. We include definitions related to the traditional TBN model from [5,15,37], the majority of which we repeat here, but please see these papers for more details and examples.

Let \( N, Z, Z^+ \) denote the set of nonnegative integers, integers, and positive integers, respectively. A key type of object in our definitions is a multiset, which we define in a few different ways as convenient. Let \( A \) be a finite set. We can define a multiset over \( A \) using the standard set notion, e.g., \( c = \{a,a,c\} \), where \( a,c \in A \). Formally, we view multiset \( c \) as a vector assigning counts to \( A \). Letting \( N^A \) denote the set of functions \( f : A \rightarrow N \), we have \( c \in N^A \). We index entries by elements of \( a \in A \), calling \( c(a) \in N \) the count of \( a \) in \( c \).

Molecular bonds with precise binding specificity are modeled abstractly as binding “domains”, designed to bind only to other specific binding domains. Formally, consider a finite set \( D \) of primary domain types. Each primary domain type \( a \in D \) is mapped to a complementary domain type (a.k.a., codomain type) denoted \( a^* \). Let \( D^* = \{a^* \mid a \in D\} \) denote the set of codomain types of \( D \). The mapping is assumed 1-1, so \( |D^*| = |D| \). We assume that a domain of primary type \( a \in D \) binds only to its corresponding complementary type \( a^* \in D^* \), and vice versa. The set \( D \cup D^* \) is the set of domain types.

We assume a finite set \( M \) of geometric monomer types \( m \in M \), where each geometric monomer type is defined as a polygon \( p \), along with a set of pairs \((d,l)\) where \( d \in D \cup D^* \) and \( l \in \mathbb{R}^2 \) is the point on the perimeter of \( p \) where \( d \) is located.

A GTBN is a pair \( T = (D,M) \) consisting of a finite set \( D \) of primary domain types and a finite set \( M \) of monomer types. Geometric monomers are taken to be rigid polygons,
and given a pair of geometric monomers, $m_1$ and $m_2$ where $(d_i, l_i) \in m_1$ and $(d_j^*, l_j) \in m_2$, if $m_1$ and $m_2$ can be positioned in the plane so that they do not overlap but the locations $l_i$ and $l_j$ on $m_1$ and $m_2$, respectively, are adjacent to each other, then those domains can bind. Bonds are rigid and therefore so are polymers formed by their binding. Geometric monomers and polymers can be translated and rotated (but not reflected), and can bind together if they can be positioned such that they do not overlap and complementary domains on their perimeters are adjacent.

In this thesis, we will only consider geometric monomers which are unit squares with at most a single domain on any face (edge), located in the center of the face. (Note that this is similar to tiles in the aTAM, but while the aTAM prevents tiles from rotating through two dimensional space, geometric monomers are allowed to within the GTBN model.) Thus, each monomer in a geometric polymer can be represented by a pair $(p, m)$ where $p \in \mathbb{N}^2$ represents the coordinates of the center of the geometric monomer and $m \in M$ the monomer type. We define a geometric polymer to be a set of such pairs and the geometric monomer binding graph to be a graph of the monomers in some GTBN with edges representing complementary domains which are adjacent to each other in some polymer.

A monomer collection $\vec{c} \in \mathbb{N}^M$ of $T$ is multiset of geometric monomer types; intuitively, $\vec{c}$ indicates how many of each monomer type from $M$ there are, but not how they are bound. A configuration is an embedding of each monomer in $\vec{c}$ into the plane $\mathbb{N}^2$, i.e. assigning each monomer a unique set of integer coordinates such that none overlap.

For any configuration $\alpha$, we can determine thermodynamic stability through the measure of two properties, enthalpy and entropy. The enthalpy of $\alpha$ is the number of bonds formed in the embedding of $\alpha$, i.e. the number of edges in the geometric monomer binding graph.
graph. The *entropy* is the number of polymers in the embedding of $\alpha$, i.e. the number of disconnect components within the geometric monomer binding graph of $\vec{c}$.

We study the particularly interesting limiting case in which enthalpy is *infinitely* more favorable than entropy. We say a configuration $\alpha$ is *saturated* if $\alpha$ has maximal bonding among all configurations in $[\vec{c}]$. We say a configuration $\alpha \in [\vec{c}]$ is *stable* if it is saturated and maximizes the entropy among all saturated configurations, i.e. every saturated configuration $\alpha' \in [\vec{c}]$ obeys $S(\alpha') \leq S(\alpha)$. We use $[\vec{c}]_{\square}$ to denote the set of stable configurations of monomer collection $\vec{c}$.

Note that, unlike traditional TBN's, due to geometric constraints, it is possible to have a configuration in a GTBN in which there exists an unbound domain $d$ on some monomer and an unbound domain $d^*$ on either that or another monomer, but $d$ and $d^*$ cannot bind together. That is, it may be impossible for the monomers (or the polymers containing them) to be validly positioned so that the domains are adjacent.

### 3.2 Efficient simulation of Turing machines in GTBN’s

In this section, we define a GTBN system that is able to simulate arbitrary Turing machines. This is formally stated in Theorem 1.

**Theorem 1.** Let $L \in DTIME(f(n))$ be a decidable language for arbitrary function $f$, and $M$ be a Turing machine which decides $L$. There exists a set of primary domain types $\mathcal{D}$, and sets of geometric monomer types $\mathcal{M}$, $\mathcal{M}_{seed}$, and $\mathcal{O} \subset \mathcal{M}$ consisting of geometric monomers with binding domains $\mathcal{D} \cup \mathcal{D}^*$ such that, for any valid input $i$ to $M$, the following properties hold:
1. there exists a set of geometric monomer types $m_i \subset M_{\text{seed}}$ such that $m_i$ collectively encodes $i$,

2. for $M_i = M \cup m_i$, there exists a geometric monomer collection $\vec{c}$ for GTBN $T_i = (D, M_i)$ such that $\vec{c}$ simulates $M$ on input $i$, and

3. the set of output geometric monomer types for the simulation is equal to $O$.

Less formally, Theorem 1 states that, for any arbitrary Turing machine $M$ that computes a decidable language $L$, there exists a set of geometric monomer types such that making a GTBN system with monomers of those types and an additional set of monomers $m_i$ that encode an input for Turing machine $M$, that system can simulate $M$ on input $i$ and produce polymers that encode the output of $M$ on input $i$ by utilizing the set of monomer types $O$.

The proof of Theorem 1 is by construction, and is similar to the proof of Theorem 1 in [15], with the creation of domains and (geometric) monomers of a GTBN $T$ based off of the definition of a zig-zag aTAM system $T_M$ which simulates the Turing machine $M$, with a few notable differences and stopping before the need to increase the size of the domain and monomer sets by creating copies hard-coded for each position in the simulation. We first note that the geometric monomers are all designed to simply be unit squares like the aTAM tiles, with single domains located in the center of faces to represent the tiles’ glues. Since this construction doesn’t require that monomers are hard-coded to locations, and in fact doesn’t require a fixed number of rows or columns, it is able to simulate a tape of steadily increasing length and so utilizes collections of monomers that combine to extend the length of the tape.

It is ensured that the deterministic path followed by the zig-zag aTAM system which
simulates the same Turing machine is faithfully encoded by the resulting “computation”
polymer of the single stable configuration by the geometric constraints placed on the po-
positioning of geometric monomers and the rigidity of their bonds, which prevents erroneous
“re-wiring” to occur as it could in the regular TBN example in Section 5 of [15]. However, in
order to create an entropy gap which makes the configuration containing the correct comput-
simulation the single stable configuration, since we can no longer have bound domains
which span the full distance of the polymer (as they do from the seed to end monomers in the
proof of Theorem 1 in [15]), we instead provide an analogous method of freeing additional
caps —thus gaining entropy— by designing the monomers so that polymers encoding the
computation combine in pairs (as seen in Figure 3.5).

Thus, an arbitrary halting Turing machine computation can be simulated efficiently
in terms of domain and monomer type counts, both of which are $O(|Q||\Gamma|)$ (where $Q$ is the
state set and $\Gamma$ is the tape alphabet). As with the construction for the proof of Theorem 1
in [15], this construction is robust over a class of configurations in which relationships exist
between the counts of different categories of monomers. The inclusion of the fact that the
language being decided $L \in DTIME(f(n))$ is simply to specify the count of computation
monomers which must be included in the collection, relative to input seeds, to ensure that the
computation can be completely represented without running out of monomers, i.e. $O(f(n)^2)$
copies of the computation monomers must be available per copy of the seed monomer.

Proof. We prove Theorem 1 by construction.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_H)$ be the Turing machine which decides $L$, with state set
$Q$, input alphabet $\Sigma$, tape alphabet $\Gamma$, transition function $\delta : (Q, \Gamma) \rightarrow (Q, \Gamma, D)$ (where $D$
Figure 3.1: This graph shows gaps in both enthalpy and entropy of different polymers in our system. The most favorable polymer should always be the paired computations.

is the set of directions \( \{L, R\} \), start state \( q_0 \), and halting state \( q_H \). For our construction, we assume that \( M \) halts by entering \( q_H \) with its tape head on a cell containing a 1 for an accepting computation or a 0 for a rejecting computation. We define the geometric monomers of \( T_i \) based on the definition of \( M \) and logically grouping them by functionality into a few main components. Note that these are logically very similar to those of the construction in Section 3 of [15], but since they are geometric monomers we must also describe their shapes and the placements of their binding domains. Also note that for easier visual depiction, the Turing machine simulations of this construction can be thought of as oriented vertically, with each successive tape and machine state encoded by a row above the previous, while the other construction was oriented horizontally.
3.2.1 Construction components

When defining specific geometric monomers, we use the convention $\text{Monomer} = \{\text{North, East, South, West}\}$. Each direction is either a domain from $\mathcal{D} \cup \mathcal{D}^*$ or corresponds to a $\lambda$ to represent that no domain is present on that side. Note that, as we said earlier, geometric monomers are allowed to rotate, meaning this convention represents an ordering of the domains as opposed to which direction they are actually pointing. The geometric monomers of $\mathcal{T}_i$ can be logically grouped into the following categories:

**Seed Monomers** We call this group of monomers $\mathcal{M}_{\text{seed}}$. This group is infinite, $|\mathcal{M}_{\text{seed}}| = \infty$, to account for the infinite unique strings that can be input into the simulated Turing machine. This group can further be broken down into two subgroups. The first being $\mathcal{M}_{\text{seed-input}}$, an infinite subgroup that contains the monomers that actually encode the input
and are the only monomers to change between different simulations of the same Turing machine. The other being $M_{seed-extra}$ which contains 9 monomers that are the same in every simulation and help the seed to operate correctly. The definitions of these monomers are given below and a visual representation is given in Figure 3.2.

$$M_{S1} = \{ EXT\_LEFT^*, R^*, seed_{1,2}, \lambda \}$$

$$M_{S2} = \{ seed_{1,2}^*, seed_{2,3}, \lambda, \lambda \}$$

$$M_{S3} = \{ _{-}^*, seed_{3,11}, \lambda, seed_{2,3}^* \}$$

$$M_{I1} = \{ [(I1), r]^*, seed_{I1,I2}, \lambda, seed_{3,11}^* \}$$

$$...$$

$$M_{I(k)} = \{ [(I_k), r]^*, seed_{I(k),I(k+1)}, \lambda, seed_{I(k-1),I(k)}^* \}$$

$$...$$

$$M_{I(n)} = \{ [(I_n), r]^*, seed_{I(n),4}, \lambda, seed_{I(n-1),I(n)}^* \}$$

$$M_{S4} = \{ _{-}^*, seed_{4,5}, \lambda, seed_{I(n),4}^* \}$$

$$M_{S5} = \{ EXT\_RIGHT^*, seed_{5,6}, \lambda, seed_{4,5}^* \}$$

$$M_{S6} = \{ \lambda, seed_{6,7}, \lambda, seed_{5,6}^* \}$$

$$M_{S7} = \{ \lambda, seed_{7,8}, \lambda, seed_{6,7}^* \}$$

$$M_{S8} = \{ \lambda, seed_{8,9}, \lambda, seed_{7,8}^* \}$$

$$M_{S9} = \{ EXT\_RIGHT, done, \lambda, seed_{8,9}^* \}$$

Note that in the monomer descriptions, any character(s) in parentheses is a place holder. Therefore, $(I1)$, $(Ik)$, and $(In)$ represent the character that those monomers encode from 0, 1, whereas $(n)$ represents the length of the input string and $(k)$ is just the number of any character $1 < k < n$. Intuitively, the seed works by doing 4 things: first, it starts the left and right extension columns which will be further discussed in the next item. Second, it pads
the input with two blank characters, which prevents the Turing machine from ever being able to run over the edge of the tape. Third, it “starts” the computation by providing the first $R^*$ domain. Lastly, it has a few monomers that attach to the right that will later connect two independent computations together to give a final entropy gap. Recall that $\vec{c}$ is defined using $M_i = M \cup \{m_i\}$ for GTBN $T_i = (D, M_i)$ where $M$ is all other monomers discussed other than $M_{seed}$ and $m_i$ is $M_{seed-extra}$ plus the monomers $M_{I(k)}$ for each $1 \leq k \leq |n|$.

**Computation Monomers** We call this set of monomers $M_{comp}$. It consists of two mutually exclusive subsets as well, the first being $M_{comp-passive}$ which consists of 6 monomers that pass values from line to line, each representing a value from the cross product of the line directions and tape alphabet, i.e. $L, R \times \_, 0, 1$. The definitions of the $M_{comp-passive}$ are below, substituting the value being passed into $(v)$:

$$M_{(v),L} = \{ [(v),r]^*, \ L, \ [(v),l], \ L^* \}$$

$$M_{(v),R} = \{ [(v),l]^*, \ R^*, \ [(v),r], \ R \}$$

The other subset is $M_{comp-transition}$ which consists of monomers that encode the logic of the transition function. The size of $M_{comp-transition}$ is asymptotically the same as the number of states times the size of the tape alphabet, i.e. $|M_{comp-transition}| = O(Q \times \Gamma)$. The construction of these monomers is such that, for ever possible $(q,s) \in Q \times \gamma$, we generate the following monomers using the specifications in the transition function $\delta(q,s) = (q', s', D)$ and $\forall v \in \Gamma$.

When $D = L$: 

26
\[ M(q,s)_{-\text{skip}} = \{ [(q, s), l]^*, R^*, [(q, s), r], R \} \]
\[ M(q,s)_{-\text{move1}} = \{ [(s'), r]^*, L, [(q, s), l], (q')^* \} \]
\[ M(q,s)_{-\text{move2}} = \{ [(q', v), r]^*, (q'), [(v), l], L^* \} \]

When \( D = R \):
\[ M(q,s)_{-\text{skip}} = \{ [(q, s), r]^*, L, [(q, s), l], L^* \} \]
\[ M(q,s)_{-\text{move1}} = \{ [(s'), l]^*, (q')^*, [(q, s), r], R \} \]
\[ M(q,s)_{-\text{move2}} = \{ [(q', v), l]^*, R^*, [(v), r], (q') \} \]

**Tape Extension Monomers** Because our goal is to simulate Turing machines that are not space-bounded, as opposed to the traditional TBN hard-coded system that simulates space-bounded Turing machines, we need additional monomers in our system that extend the tape of the Turing machine. We define this group of eight monomer types to be \( \mathcal{M}_{\text{ext}} = \{ M_{L1}, M_{L2}, M_{L3}, M_{L4}, M_{R1}, M_{R2}, M_{R3}, M_{R4} \} \).

\[ M_{L1} = \{ \text{extL}_{4,1}^*, \text{extL}_{1,2}, \lambda, \lambda \} \]
\[ M_{L2} = \{ \text{extL}_{2,3}, L, \text{EXT}_{-LEFT}, \text{extL}_{1,2}^* \} \]
\[ M_{L3} = \{ [\_\_, l]^*, R^*, \text{extL}_{2,3}^*, \text{extL}_{3,4} \} \]
\[ M_{L4} = \{ \text{EXT}_{-LEFT}^*, \text{extL}_{3,4}^*, \text{extL}_{4,1}, \lambda \} \]
\[ M_{R1} = \{ \text{extR}_{4,1}, \lambda, \lambda, \text{extR}_{1,2} \} \]
\[ M_{R2} = \{ \text{extR}_{2,3}, \text{extR}_{1,2}^*, \text{EXT}_{-RIGHT}, R \} \]
\[ M_{R3} = \{ [\_\_, r]^*, \text{extR}_{3,4}, \text{extR}_{2,3}^*, L^* \} \]
\[ M_{R4} = \{ \text{EXT}_{-RIGHT}^*, \lambda, \text{extR}_{4,1}, \text{extR}_{3,4}^* \} \]

These monomers don’t encode values, just blank spaces that will pass up to higher rows to potentially be used in the Turing machine simulation. By utilizing these monomers,
we ensure that the tape is always big enough for the computation to continue. An example of these supertiles is shown in Figure 3.3b.

**End Monomers** We call this group of monomers $M_{\text{end}}$. These monomers only connect to the computation after the simulated TM goes into a halting state and act similar to the monomers in $M_{\text{comp-passive}}$ while also passing a signal that the computation is finished. The monomers will form one row if passing above the halting state $M_{(q_H),(v)}$ to the right or two rows if passing above the halting state to the left. This will ensure that the top row of the final computation monomer is a right-growing row, which is necessary for the eventual pairing of two complete computations. It is this final row that is input into the encoding function $E_{\text{output}}$ to read the result of the TM.

For all halting states $(q_H)$ and output characters $(v) \in 0, 1$, should the halting state first be read with a left passing row, it attaches the following monomer:

$$M_{(v),\text{haltL}} = \{ [(q_H,v),r]^*, L, [(q_H),(v),l], L^* \}$$

and on the subsequent right passing row, it will attach one of the following monomers, the same monomer that is checked for as input to the $E_{\text{output}}$ function:

$$M_{(v),\text{haltR}} = \{ \lambda \ halt^*, [(q_H,v),r], R \}$$

and lastly, it will attach a string of the following monomers to finish off the top row, the last of which will also attach to the seed of the paired computation:

$$M_{(v),R1} = \{ \lambda, \ halt^*, [(v),r], halt \}$$
**Capping Monomers**: Caps in this system are very important because they prevent unwanted polymers from forming and make it easier to argue about the outcome of the system. Each of the monomer types talked about previously, the seed monomers, computation monomers, tape extension monomers, and end monomers all have caps. Each cap consists of three monomers, which form an "L" shape to reach both inputs of the capped monomer. The caps all have codomains $x^* \in D^*$, since every input in the system is from the set of domains $x \in D$, which prevents them from sticking together. While there are too many caps to explicitly define each individually, we do give an example of the seed cap:

$M_{seed-cap1} = \{ \lambda, seedcap_{1,2}, EXT\_RIGHT^*, \lambda \}$

$M_{seed-cap2} = \{ \lambda, \lambda, seedcap_{2,3}, seedcap_{1,2}^* \}$

$M_{seed-cap3} = \{ seedcap_{2,3}^*, \lambda, \lambda, halt^* \}$

All other caps follow this same convention, just replacing $seedcap_{1,2}$ and $seedcap_{2,3}$ with other unique glues and $halt^*$ and $EXT\_RIGHT^*$ with the codomains that correspond to the inputs of the monomer to be capped.

There are two key dynamics our system uses, the first of which is the formation of **supertiles**. A supertile in our system is a set of monomers that have a unique set of matching domains/codomains between them but aren’t capped with respect to one another. Therefore, it is always favorable to have them bind together, regardless of how other monomers in the system bind. The supertile itself can be capped, however, and all caps in this system are actually supertiles themselves as well. The other supertiles in our construction are the seed and tape extension pieces. Supertiles allows us to utilize the properties of polyominoes in our system while only requiring unit square shaped monomers. To simplify our arguments, we assume that the counts of all the monomers in the same supertile to be the same, as to
avoid dealing with partially formed supertiles. Therefore, the counts of every monomer type from $M_{\text{seed}}$ in $\tilde{c}$ are equal, the counts of every monomer type from $M_{\text{ext}}$ in $\tilde{c}$ are equal, and the counts of every monomer type that makes up one cap are equal. Note that this assumption is analogous to having polyominoes replace supertiles in our system, which we avoided because we felt unit square geometric monomers were the TBN model that most naturally corresponded to the traditional aTAM model.

The other key dynamic is a fully capped state. We utilize one other assumption in our system, that the count of each cap $x$ in the system is always greater than the number of monomers or supertiles that $x$ corresponds to. We use the term fully capped state to refer to a configuration of the system in which every monomer or supertile that has a cap is bound to that cap through both domains. The individual monomers that bind supertiles are still bound in this state too, but no other polymers should be bound together. We use this state as a reference, somewhat of a neutral baseline, to show how correct computations are favored and all other non-capped polymers are unfavored.

In addition to the geometric monomers of $T$, we give relative counts for the monomers in $\tilde{c}$. This mean our system does not rely on exact counts of each monomer, but rather a group of assumptions about the relative counts. To show that every seed supertile is part of a complete paired computation polymer, we require that the number of cap supertiles in $\tilde{c}$ is greater than the number of computation monomers, tape extension supertiles, and end monomers, whose counts in $\tilde{c}$ are greater than the number of seed supertiles by a factor of $O(f(n)^2)$ (where $L \in \text{DTIME}(f(n))$ is the language being decided by the Turing machine $M$). In other words, we rely on the assumptions that: (1) every pair of seed supertiles has enough computation monomers, tape extension supertiles and end monomers to create a
complete paired computation polymer (which is the reason for needing $O(f(n)^2)$ of those types relative to seed types), and (2) every computation monomer, tape extension supertile, and end monomer can have its own cap if it wasn’t bound to anything else.

One last bit of terminology that we use in the following proofs is *inputs* and *outputs*. By inputs, we are referring to the two domains (not codomains) on the seed, computation monomers, tape extension supertiles, and end monomers. These two domains on each monomer/supertile mentioned are the domains that can bind to the associated cap. By outputs, we are referring to all of the codomains (not domains) on the seed, computation monomers, tape extension supertiles, end monomers and capping supertiles. While all computation monomers have two outputs, the seed can have any number, tape extension supertiles have three, and end monomers can have one or two. Notice that outputs only bind to inputs, and inputs only bind to outputs. The only domains present in our system that are not inputs or outputs are the unique domains that bind supertiles together.

**Claim 3.2.1.** The fully capped state is saturated.

*Proof.* The argument for this claim is simple. Recall that a system is saturated if there are no labels in which both a domain and a codomain are left unbound. Any element $x \in D$ in our system falls into two categories, $D_{\text{supertiles}}$ which are unique domains used to form supertiles, and $D_{\text{capped}}$ which are input or output domains used to encode states, values, signals, etc. and always have a cap. In the fully capped state (and all other configurations), supertile monomers are always bound together forming the maximum number of supertiles and ensuring as many domains in $D_{\text{supertiles}}$ are bound as possible. For domains in $D_{\text{capped}}$, because the number of complete cap supertiles for each pair of inputs $(I_1, I_2)$ in $\bar{c}$ is greater
than the number of monomers or supertiles of the type that corresponds to those inputs, every domain in $\mathcal{D}_{capped}$ will also be in a bond. Therefore, no domain in the system will be left unbound, meaning that the fully capped state of $\vec{c}$ is saturated.

3.2.2 Desired configuration

![Diagram showing desired configuration](image)

**Figure 3.3:** The addition of all correct pieces to the seed are enthalpy and entropy neutral. Each breaks two bonds with the cap and forms two more with the computation while also remaining at two polymers before and after the transition.

We now argue about the desired configuration of $\vec{c}_i$, which we call $\alpha$, that consists of 5 types of polymers. These are (1) polymers containing two instances of the seed supertile, computation monomers corresponding to two correct and complete computations of $M$ on input $i$, a series of left tape extension supertiles and a series of right tape extension supertiles, and one final partial row of end monomers that, along with the final right tape extension supertiles, connect to one instance of the seed, (2) polymers that consist of a single capping supertile bound to a single computation monomer and bound by 2 binding domains, (3) polymers that consist of a single capping supertile bound to a single tape extension supertile
and bound by 2 binding domains, (4) polymers that consist of a single capping supertile bound to a single end monomer and bound by 2 binding domains, and (5) polymers that are single capping supertiles. For polymers described by (1), we let \( n \) denote the number of computation monomers, tape extension supertiles, and end monomers in such a polymer plus one. We will prove that \( \alpha \) is saturated, and then show that \( \alpha \) is the only stable configuration of \( \mathcal{C}_i \).

**Claim 3.2.2.** \( \alpha \) is saturated.

**Proof.** Similar to the proof of Claim C.1 in [15], we define a sequence of configurations \( \alpha_i \in \text{seq}(\alpha) \) for \( 0 \leq i \leq n \) where \( \alpha_0 \) is the fully capped state, \( \alpha_n = \alpha \), and for every \( i > 1 \), \( \alpha_i \) is saturated if \( \alpha_{i-1} \) is saturated. Furthermore, we also define \( p(\alpha_i) \) to be the polymer that contains the seed(s) in the configuration \( \alpha_i \) and \( p(\text{seq}(\alpha)) \) to be the set of all polymers \( p(\alpha_i) \) for \( 0 \leq i \leq n \), i.e. a seed supertile, the complete paired computation, and everything in between. The difference between every \( \alpha_{i-1} \) and \( \alpha_i \) is the transition of just one computation monomer, tape extension supertile, or end monomer from a polymer with its cap to the polymer \( p(\alpha_i) \). Notice at any point in the sequence before \( \alpha_n \) there will always be one (and only one) concave corner at the end of the partially formed top row of \( p(\alpha_i) \), where two unbound output domains will be adjacent to the same unit square location. Because of the deterministic nature of \( M \), there is exactly one type of computation monomer, tape extension supertile, or end monomer (or seed when \( i = n - 1 \)) with input domains that correspond to the unbounded output domains at this location. The piece that corresponds to this input is the piece that transitions from its cap to \( p(\alpha_i) \) in the step \( \alpha_{i-1} \rightarrow \alpha_i \). The only exception is the final step \( \alpha_{n-1} \rightarrow \alpha_n \) which is the pairing of two complete computation
polymers together. This step works by starting with two independent computations polymers that have completed their respective instances of the simulation and have one final concave corner location with the unbound output domains \( \text{halt}^* \) and \( \text{EXT\_RIGHT}^* \) exposed. These correspond to the capped input domains on the far right of the seed supertile. In this step \( \alpha_{n-1} \rightarrow \alpha_n \), both seeds break off their respective cap and bind to the unbound output domains on the other computation polymer.

We view the enthalpy and entropy difference of each step \( \alpha_{i-1} \rightarrow \alpha_i \) as independent of the rest of the system, considering the rest of the system is constant while the transition is happening. From Figure 3.3, it is easy to see how the transition of computation monomers, tape extension pieces, and end monomers from their respective caps to \( p(\alpha_i) \) is always enthalpy and entropy neutral. Each transition breaks two bonds and makes two more bonds and starts with two polymers and ends with two polymers. Fast forwarding to the end of the sequence, it is also clear from Figure 3.4 that the overall enthalpy and entropy of the \( \alpha_{n-1} \) configuration is the same as the overall enthalpy and entropy of \( \alpha_0 \) (in this example, 86 bonds and 43 polymers). The last step of the sequence, illustrated in Figure 3.5, shows how the pairing of two complete computation polymers is enthalpy neutral (breaking 4 bonds and making 4 bonds) but actually gives an entropy bonus of +1 to the system (starting with 2 polymers and ending with 3 polymers). Because this final configuration has the same enthalpy as the fully capped state, which is saturated, we know this configuration is saturated as well.

\( \square \)

**Claim 3.2.3.** A polymer in \( \alpha_n \) that contains two seeds will contain two monomers of the
type $M(v)_R$ that encode the same value of $v$.

Proof. First, notice the south input to the $M(v)_R$ monomer is the $[(q_H, v), r]$ input domain. The corresponding output domain $[(q_H, v), r]^*$ is only present on the north face of the $M_{(q,s)-move2} \in M_{comp-transition}$ monomer. However, the $M_{comp-transition}$ set is designed so that every monomer in the set can only bind one of its input domains to one of the output domains of another monomer in the set or the seed. Therefore, the seed starts one path of monomers exclusively in $M_{comp-transition}$. This path ends when either $M(v)_L$ or $M(v)_R$ is attached. If $M(v)_L$ is attached first, it subsequently attaches one monomer of the $M(v)_R$ type. Because this path is started by the seed and the main polymer of $\alpha$ has two seeds, this polymer will also contain two monomers of the type $M(v)_R$.

Because the simulated Turing machine is deterministic, every step from $\alpha_i$ to $\alpha_{i+1}$ has a unique monomer or supertile making the transition from its cap to the computation polymer. Therefore, the two independent computations that are paired together in the main polymer of $\alpha$ must be identical, therefore ensuring that the two monomers of type $M(v)_R$ encode the same value $v$.

3.2.3 Incorrect computations

We already showed that $\alpha$ is saturated. In order to show that $\alpha$ is stable, we need to show that all other saturated configurations in $\vec{c}$ have less entropy. We start by showing the majority of configurations in $\vec{c}$ aren’t saturated at all. Then we show the few that are also saturated create less polymers than $\alpha$.

First, we prove claims that help us show that configurations not in $seq(\alpha)$ are unstable.
Figure 3.4: On the left is a completed computation that has not yet been paired. Compared to the fully capped state, it has the same number of bonds and polymers, meaning it still has neutral enthalpy and entropy.

Claim 3.2.4. Assuming an excess of caps, any configuration of $\vec{c}$ that contains at least one monomer/supertile with an unbound input is not saturated.

Proof. Given an excess of caps, the fully capped state is possible. Since the fully capped state has every input of all monomers and supertiles in the $\vec{c}$ bound, and since inputs are the limiting factor on the maximum number of bonds in $\vec{c}$, then any configuration that does not have every input bound has not formed the maximum number of bonds and is therefore unsaturated.

Claim 3.2.5. Assuming an excess of caps, any configuration $\vec{c}$ that includes a polymer $p$ consisting of 2 or more caps is not stable.

Proof. This only requires a simple counting proof. Assume $p$ consists of $M$ monomers and supertiles (each supertile counts as one) and $N$ caps. Considering the caps of the $M$ monomers and supertiles, this accounts for $M + 1$ polymers. However, in a fully capped state, $p$ could be broken up with its caps into $M + N$ polymers. Therefore, whenever $N > 1$, the number...
Figure 3.5: This illustrates the pairing of two complete computations. The dislodging of the seed caps creates an entropy bonus of one. This bonus is what makes the final complete paired construction favorable.

of polymers in the fully capped state, $M + N$, is greater the number of polymers in $\vec{c}$, $M + 1$, meaning $\vec{c}$ is not stable.

Claim 3.2.6. If any two monomers $m1$ and $m2$ in a single polymer are connected through a path of other monomers and supertiles that does not include the seed, $m1$ and $m2$ must have the same rotation.

Proof. Although rotation of monomers and polymers relative to each other is permitted in the GTBN model, we design domains so that they are only complementary in north/south or east/west pairs, except for the seed/end monomer connection and the seed/right tape extension supertile connection that allow paired computations to combine. Therefore, if two monomers are bound together in a polymer of $\vec{c}$ by a path that does not contain the seed, one cannot be rotated relative to the other.

We are also going to make use of the idea of a subpolymer. We define a subpolymer of a polymer $p$ to be another polymer made up of a set of ordered pairs of monomer types and corresponding locations that is a subset of the set of ordered pairs in $p$, i.e. it is the same polymer as $p$ but with some monomers potentially missing. Additional terminology
that we will use in these proofs is \textit{signal}, \textit{pumping}, and \textit{stopper}. These proofs are all focused on finding an unbound input in a polymer. By signal, we are referring to any domain that has an input on one face of a monomer and the corresponding output on the opposite face. Examples of signals are \(L, R, \text{EXT\_LEFT}, \text{EXT\_RIGHT}, \) and \textit{halt}. Pumping is the idea that trying to bind an unbound input instance of a signal with one of these monomers will just expose another input instance. Finally, a stopper is a monomer or supertile that can bind to an unbound input instance of a signal without exposing another input instance, thereby stopping the signal from pumping. For example, the left tape extension supertile acts as a stopper for the \(R\) signal because it has an \(R^*\) output domain but does not have an \(R\) input domain. Likewise, the seed is a stopper for the \textit{EXT\_LEFT} signal because it has an \textit{EXT\_LEFT}^* output domain but does not have an \textit{EXT\_LEFT} input domain.

Claim 3.2.7. Any configuration of \(\vec{c}\) that includes a polymer \(p_{\text{incorrect}}\) that contains the seed but is not in the set \(p(seq(\alpha))\) is unsaturated.

\textit{Proof.} This proof utilizes the fact that our TM is deterministic, and therefore, every pair of inputs \((I_1, I_2)\) where \(I_1, I_2 \in \mathcal{D}\) corresponds to one unique monomer. Take that polymer \(p_{\text{incorrect}}\) that contains the seed but differs from any polymer in \(p(seq(\alpha))\). Let \(p_{\text{sub}}\) be the largest polymer such that \(p_{\text{sub}} \in p(seq(\alpha))\) and \(p_{\text{sub}}\) is a subpolymer of \(p_{\text{incorrect}}\). Notice that such a polymer must exist, since \(p(\alpha_0)\) is the seed and we are only considering polymers that contain the seed. Also notice that all polymers in \(p(seq(\alpha))\) will only ever have 5 unique types of output codomains (and 0 input domains) unbound and exposed: \(\text{EXT\_LEFT}^*\), \([(v),\{l, r\}]^*\), \([(q', v'),\{l, r\}]^*\), \(\text{EXT\_RIGHT}^*\), and one of the three signal output domains \{\(L^*, R^*, \text{halt}^*\}\). Since \(p_{\text{incorrect}}\) is different than any polymer in \(p(seq(\alpha))\), then at least one
of these output domains extending from \( p_{\text{sub}} \) must be bound to a monomer or supertile that is different than the monomer or supertile that it is bound to in \( \alpha \).

In the first case, assume the incorrect extension was bound to \( p_{\text{sub}} \) by the output domains of either \( \text{EXT}_\text{LEFT}^* \) or \( \text{EXT}_\text{RIGHT}^* \). Since the corresponding input domains \( \text{EXT}_\text{LEFT} \) and \( \text{EXT}_\text{RIGHT} \) are unique to the tape extension supertiles, these output domains cannot attach the incorrect piece. The only other mistake that can happen is if a tape extension supertile attaches past the final row of computation. Assuming this had happened with a right supertile, this would expose an \( R \) input domain. This input could either be unbound, causing the configuration to be unsaturated, or it could be attached to another seed, a computation monomer, or a left tape extension supertile, the three pieces with \( R^* \) output domain. It’s easy to see another seed or a left tape extension supertile would be blocked geometrically from attaching here. A computation monomer could be attached, however, it would have to pump the \( R \) signal to avoid an unbound input. There are two subcases that could happen. One, it would reach the edge of a partially formed top row of \( p_{\text{sub}} \). Here, it could be attached to a computation monomer that was hanging off the edge, but this computation monomer would have an input to the south that was unbound. If the unbound input was also bound by a computation monomer, that computation monomer would be the correct piece, thereby contradicting that \( p_{\text{sub}} \) is maximal. The hanging monomer could also have a cap, which would cause \( p_{\text{incorrect}} \) to have another cap in addition to the seed cap, making it unstable by Claim 3.2.5. Instead of additional computation monomers, the last computation monomer of the pumping chain could also be attached to a left tape extension supertile. However, since the inputs of the supertile would be pressed against \( p_{\text{sub}} \), the inputs would have to be connected to outputs of \( p_{\text{sub}} \). If this was the case, the
supertile would be correct, contradicting that \( p_{sub} \) is maximal. Two, it would pump the
signal to a location that would require a monomer that has the input domain \([q_H, v), r]\) on
its south face and output domain \(R^*\) on its east face, of which there are none. Assuming
the extra tape extension supertile was on the left, the supertile would require an \( L \) input
domain to be bound. This would yield two symmetrical subcases. One, the signal would
pump with computation monomers to the edge of a partially formed top row of \( p_{sub} \), which
we just showed leads to the configuration being unstable. The signal could also pump until
it reached the location north of \( M(v, haltR) \) monomer. Because the \( M(v, haltR) \) monomer has no
output on the north face and all computation pieces have an input on the south face, no
monomer could be in this location and have its south input bound, meaning there must be
an unbound input somewhere, making the configuration unsaturated.

In the second case, \([v, \{l, r\}]^* \) and \([(q', v'), \{l, r\}]^* \) can only attach a computation
monomer or end monomer. Because these domains encode the direction of the next row,
any attached computation or end monomer must also have an input domain in the opposite
direction. In other words, any monomer attached to \([v, l]^* \) or \([(q', v'), l]^* \) must have an
input domain \( L \) on the east face, likewise with \([v, r]^* \) or \([(q', v'), r]^* \) and an input domain
\( R \) (or \( halt \)) on the west face. In order to have this input also bound, there must be another
computation monomer or end monomer connected to this input, with another \( L \) or \( R \) (or
\( halt \)) on its east or west face, respectively. This signal would have to pump to avoid an
unbound input. There’s two possibilities for the end of this chain: 1) it eventually connects
to the signal output domain \( \{L^*, R^*, halt^*\} \) on \( p_{sub} \). If this happens, the last monomer
would have two input domains that match two output domains from \( p_{sub} \), meaning it was
the correct tile, contradicting that \( p_{sub} \) was maximal. 2) the chain of monomers would pump
to the edge of the partially formed top row of $p_{sub}$. Similar again to the argument in case one, the input of the last monomer in the chain could either be attached to: (a) another computation monomer hanging over the edge or (b) a tape extension supertile. For (a), the hanging computation monomer would require another computation monomer to the south, a bound cap, or would have an unbound input domain. Another computation monomer to the south would have to match two input domains to $p_{sub}$, contradicting that $p_{sub}$ was maximal. A bound cap would be the second on $p_{incorrect}$, in addition to the seed cap, making the configuration unstable by Claim 3.2.5. For (b), the supertile would have its inputs pressed against $p_{sub}$, forcing it to bind to $p_{sub}$ to avoid an unbound input. This, however, would make the supertile the correct piece for that location, contradicting that $p_{sub}$ was maximal.

In the last case, if any incorrect monomers or supertiles are bound to the signal output domain $\{L^*, R^*, \text{halt}^*\}$, that monomer or supertile must also have an input domain to the south that binds with $p_{sub}$. However, if this monomer or supertile matches both of its input domains to two output domains of $p_{sub}$, then it must correspond to the correct monomer or supertile for that location, once again contradicting that $p_{sub}$ was maximal. These three cases show that any erroneous addition to a polymer in $p(seq(\alpha))$ make the configuration containing that polymer unsaturated, and thus unstable.

3.2.4 Seedless polymers

Now we will show that configurations in $\tilde{c}$ that include polymers that do not contain the seed and do contain more than one element from the set of all computation monomers, tape extension supertiles, and end monomers are unstable. We start by proving a few claims
regarding signals.

**Claim 3.2.8.** If a polymer does not contain the seed and does contain a monomer with the input domain $EXT\_LEFT$, it has either an exposed instance of the $EXT\_LEFT$ domain or an attached cap.

*Proof.* First, notice that the only monomer or supertile that contains the $EXT\_LEFT$ domain as an output is the left tape extension supertile. However, this supertile also has the domain as an input. Therefore, if a polymer contains the $EXT\_LEFT$ input domain, it can either be exposed, capped, or bound to an instance of the tape extension supertile. The last case is the only implication not included in the claim, but it exposes another $EXT\_LEFT$ input domain, causing the signal to pump. Whereas, in polymers with the seed, the seed acts as the stopper for this signal, but without the seed, no other monomers or supertile can act as a stopper. Because polymers cannot be infinite, eventually one of these input domains will have to be left exposed or capped. Note that this proof is analogous for $EXT\_RIGHT$. □

**Claim 3.2.9.** If a polymer does not contain the seed and does contain a monomer with input domain $R$, it has either an exposed instance of the $R$ domain, an exposed instance of the $EXT\_LEFT$ domain, or an attached cap.

*Proof.* Using the same reasoning as the proof for Claim 3.2.8, notice that the only monomers and supertiles that contain the $R$ domain as an output also have the domain as an input. The only exception this time is the left tape extension supertile, which does have $R^*$ as an output and does not have $R$ as an input. Therefore, if a polymer contains the $R$ input domain, it can either be exposed, capped, bound to a left tape extension supertile, or bound to another monomer that also has $R$ as an input domain. The first two cases are implications
in the claim, whereas the third case, being bound to a left tape extension supertile, exposes an
*EXT_LEFT* input domain, which also leads to the implications of Claim 3.2.9 using Claim
3.2.8. The final case, being bound to another monomer that also has *R* as an input domain,
creates a new instance of the *R* input domain causing the signal to pump. Because polymers
cannot be infinite, eventually one of the first three cases will happen, eventually leading to
an exposed input domain or a cap. Note that this proof shows \( R \rightarrow \text{EXT._LEFT} \) and is
analogous for \( \text{halt} \rightarrow R \) and for \( L \rightarrow \text{EXT._RIGHT} \).

**Claim 3.2.10.** Any configuration of \( \vec{c} \) that includes a polymer \( p_{\text{seedless}} \) that does not contain
the seed but does contain two or more elements from the set off all computation monomers,
tape extension supertiles, and end monomers is unstable.

**Proof.** We’re going to break up this proof into three cases. In the first case, \( p_{\text{seedless}} \) has
a tape extension supertile. Without loss of generality, assume it is a left tape extension
supertile. This supertile has two inputs, *EXT_LEFT* and *L*. By Claim 3.2.8, *EXT_LEFT*
implies there is an unbound input domain or an attached cap, and by Claim 3.2.9, *L* implies
there is another unbound input domain or attached cap that is independent of the first. Since
\( p_{\text{seedless}} \) must have at least one unbound input or two attached caps, \( p_{\text{seedless}} \) is unstable.

In the second case, \( p_{\text{seedless}} \) has one *L* input domain and one *R* (or *halt*) input domain,
or two of the same input domain on different rows, i.e. connected through north-south do-
mains. If the signals are different, Claim 3.2.9 can apply to both, indicating that \( p_{\text{seedless}} \) has
at least one exposed input or two attached caps. If they are the same signal on different rows,
there is the possibility that they could turn into the same *EXT_LEFT* or *EXT_RIGHT*
signal. However, if they did, \( p_{\text{seedless}} \) must contain at least one tape extension supertile,
making this case one again which we already showed is unstable.

The only other case is when $p_{seedless}$ is a single row of computation (or end) monomers, i.e. a polymer with just one $L$ or $R$ (or $halt$) signal. However, since these polymers consist of computation (or end) monomers, each of which has a second input on its south face, the overall polymer will still have input domains, in the form of values from another row, that need to be addressed. If these input domains are left unbound, then the configuration isn’t maximizing bonds and is therefore unsaturated. If these domains are bound, it must be with additional computation monomers, which will create another instance of an $L$ or $R$ in another row, giving the polymer two signals which we already showed is unstable. One important note is that, if the original $L$ or $R$ (or $halt$) signal is capped, it will also cap one of these value input domains. However, since we are arguing about polymers with at least two non-cap monomers or supertiles, there must be at least one other value input domain, which can only be bound by the previously mentioned additional row of computation monomers. Examples of unstable spurious computations are shown in 3.6.

To summarize the previous section, we have shown that all configurations in $seq(\alpha)$ are saturated, from the $\alpha_0$, the fully capped state, to $\alpha$, the configuration in which every seed is bound into a complete paired computation. We also showed that the final configuration $\alpha$ has a higher entropy than every other configuration in $seq(\alpha)$. Then we showed that any configuration with a polymer that contains the seed but is not an element of $p(seq(\alpha))$ has unbound input domains and is therefore not saturated. Finally, we showed that any polymer that does not contain the seed and does contain at least two elements from the set
Figure 3.6: On the left is an example of a spurious computation with exposed domains. Because these domains are unbounded, there is an enthalpy gap between this polymer with its detached caps and a fully capped state. On the right is a special case where all the exposed input domains are still bound to their caps. In this case, there is an entropy gap with a fully capped state. To clarify, these two subsystems are not being compared to each other, but the statistics below each image are the comparisons between each polymer and a fully capped state.

of all computation monomers, tape extension supertiles, end monomers, and caps is either unsaturated or has a lower entropy than $\alpha_0$, the fully capped state. The accumulation of all these results proves that $\alpha$ is the single stable configuration of $\vec{c}$.

Now that we have shown how $\vec{c}$ simulates $M$, that $p(\alpha)$ will always contain two matching monomers of type $M_{(v),\text{haltR}}$, and proved that $\alpha$ is the stable configuration of $\vec{c}$, all that is left to do is to show how $E_{\text{output}}$ works. $E_{\text{output}}$ works by looking at the stable configuration of $\vec{c}$, analyzing the definition of any polymer that contains the seed supertile ($p(\alpha)$), and outputs a 0 if that polymer contains two instances of $M_{0,\text{haltR}}$ or 1 if that polymer contains two instance of $M_{1,\text{haltR}}$. The proof that two monomers of one of these two types will be in every instance of the final complete paired computation polymer is a combination of the proof that $p(\alpha)$ contains these two monomers and the proof that $\alpha$ is the stable configuration. This final output 0 or 1 corresponds to the output of the Turing machine $M$.  

\qed
4 Utilizing 2D Components to Build 3D Structures

The next model we investigate with respect to the abstract Tile Assembly Model is the Flexible Tile Assembly Model (FTAM) [38] [16]. We introduced this model in [38] and refined in [16]. This model is a generalization of the aTAM by allowing the bonds between tiles to be either rigid or flexible. Tiles connected through flexible bonds are allowed to flex out of the 2D plane, thereby creating 3D structures. One motivation behind this new model is that it gives us a foundation in studying how local interactions within a tile assembly system can affect the global orientation of the system. This can be useful in modeling natural processes, such as the folding of a string of amino acids into a protein. Additionally, this model is motivated by evidence that it can be physically realizable. Whenever tiles are implemented in physical labs, the bonds between tiles can be designed to include portions of single stranded DNA. Since single stranded DNA doesn’t have the rigid double helix shape of double stranded DNA, this essentially allows the bonds to flex and the tiles to move in relation to one another, similar to the dynamics that we will explore here. In this chapter, we will first define the model, and then we will explore the ability of systems within the model to create 3D structures.

4.1 Formal definition of the Flexible Tile Assembly Model

Here we present definitions related to the FTAM. Note that, while the ideas of tiles, the assembly process, and more are similar to the ones we defined in Section 2.2, we redefine them here to adhere to the new environment and dynamics of the FTAM.
A tile type $t$ in the FTAM is defined as a 2D unit square that can be translated, rotated, and reflected throughout 3-dimensional space, but can only occupy a location such that its corners are positioned on four adjacent, coplanar points in $\mathbb{Z}^3$. Each tile type $t$ has four sides $i \in \{N, E, S, W\}$, which we refer to as $t_i$. Let $\Sigma$ be an alphabet of labels and $\bar{\Sigma} = \{a^*|a \in \Sigma\}$ be the alphabet of *complementary labels*, then each side of each tile has a *glue* that consists of a *label* $\text{label}(t_i) \in \Sigma \cup \bar{\Sigma} \cup \epsilon$ (where $\epsilon$ is the unique *empty* label for the *null glue*), a non-negative integer *strength* $\text{str}(t_i)$, and a boolean valued *flexibility* $\text{flx}(t_i)$.

A tile is an instance of a tile type. A *placement* of a tile $p = (l, n, o)$ consists of a location $l \in \mathbb{Z}^3$, a *normal* vector $n$ which starts at the center of the tile and points perpendicular to the plane in which the tile lies (i.e. $n \in \{+x, -x, +y, -y, +z, -z\}$\(^1\)), and an *orientation* $o$ which is a vector lying in the same plane as the tile which starts at the center of the tile and points to the $N$ side of the tile (i.e. $o \in \{+x, -x, +y, -y, +z, -z\}$). Note that by convention, to avoid duplicate location specifiers for a given tile, we restrict a location $l$ to refer to only the 3 possible tile locations with corners at $l$ and which extend in positive directions from $l$ along one of the planes (i.e. tiles are located by their vertices with the smallest valued coordinates). For any given $l$, there can only be a max of one tile with $n \in \{+x, -x\}$, one tile with $n \in \{+y, -y\}$, and one tile with $n \in \{+z, -z\}$.

Let $p = (l, n, o)$ and $p' = (l', n', o')$ be placements of tiles $t$ and $t'$, respectively, such that $p$ and $p'$ are non-overlapping and for some $i, j \in \{N, E, S, W\}$, sides $t_i$ and $t'_j$ are adjacent (i.e. touching). We say that $p$ and $p'$, have *compatible* normal vectors if and only if either (1) $n = n'$, (2) $n$ and $n'$ intersect, or (3) $\text{inverse}(n)$ and $\text{inverse}(n')$ intersect.\(^2\)

\(^1\)We refer to the vectors $\{(1, 0, 0), (-1, 0, 0), (0, 1, 0), (0, -1, 0), (0, 0, 1), (0, 0, -1)\}$ by the shorthand notation $\{+x, -x, +y, -y, +z, -z\}$ throughout this thesis.

\(^2\)The *inverse* function simply negates the signs of the non-zero components of a vector.
(See Figures 4.1a and 4.1b.) We will refer to these three orientations as “Straight”, “Up”, and “Down”, respectively. Furthermore, if (1) label($t_i$) is complementary to label($t'_j$), (2) $str(t_i) == str(t'_j)$, (3) $flx(t_i) == False$ and $flx(t'_j) == False$, and (4) $n$ and $n'$ are in a “Straight” orientation, then the glues on $t_i$ and $t'_j$ can bind with strength value $str(t_i)$ to form a rigid bond. Similarly, if (1) label($t_i$) is complementary to label($t'_j$), (2) $str(t_i) == str(t'_j)$, (3) $flx(t_i) == True$ and $flx(t'_j) == True$, and (4) $n$ and $n'$ are compatible, then the glues on $t_i$ and $t'_j$ can bind with strength value $str(t_i)$ to form a flexible bond.  

Figure 4.1: Possible normal vectors of pairs of tiles. Those in (a) are compatible and allow a bond to form between complementary glues in the orientations “Up”, “Down”, and “Straight”, respectively. Those in (b) are not compatible.

We define an assembly $\alpha$ as a graph whose nodes, denoted $V(\alpha)$, are tiles and whose edges, denoted $E(\alpha)$, represent bound complementary glues between adjacent edges of two tiles. An edge between sides $i$ and $j$ of tiles $t$ and $t'$, respectively, is represented by the tuple $(t_i, t'_j)$, which specifies which sides of $t$ and $t'$ the bond is between. Whether or not it is flexible can be obtained by $flx(t_i)$ and its strength can be obtained by $str(t_i)$ (since those values must be equal for both $t_i$ and $t'_j$).

---

3Note that any glue can only bind to a single other glue, and also that if edges of 4 tiles are all adjacent to each other, if glues of 2 tiles which are co-planar bind, then that “blocks” any possible binding between the other pair (which must be co-planar to each other) since that bond would have to cross through the existing bond.
We define a face to be a set of coplanar tiles that are all bound together through rigid bonds. Additionally, we define a face graph to be a graph minor of the assembly graph where every maximal subgraph in which every node can be reached from every other node using a path of rigid tiles is replaced by a single node in the face graph. Two nodes in the face graph that correspond to two groups of nodes in the assembly graph have an edge if and only if there is at least one flexible bond between any single node in the first group of the assembly graph and any single node in the second group of the assembly graph. Conversely, the assembly graph is an inflation of the face graph.

An FTAM system is a triple $T = (T, \sigma, \tau)$ where $T$ is a finite set of tile types (i.e. tile set), $\sigma$ is an initial seed assembly, and $\tau \in \mathbb{Z}^+$ is a positive integer which specifies the minimum binding threshold for tiles and is referred to as the temperature parameter. An assembly is $\tau$-stable if and only if every cut of edges of $\alpha$ which separates $\alpha$ into two or more components must cut edges whose strengths sum to $\geq \tau$. We will only consider assemblies which are $\tau$-stable (for a given $\tau$), and we use the term assembly to refer to a $\tau$-stable assembly.

Given an assembly $\alpha$, a configuration $c_\alpha$ is a mapping from every flexible bond in $\alpha$ to an orientation from \{“Up”, “Down”, “Straight”\}. An embedding $e_\alpha$ is a mapping from each tile in $\alpha$ to a placement. Given an assembly and a configuration, we can obtain an embedding by choosing any single initial tile and assigning it a placement and computing the placement of each additional tile according to how it is bonded with tiles that are already placed. Note that, given tiles to which it is bound, their placements, and an orientation, there is only one tile location at which each additional tile can be placed. We say a configuration $c_\alpha$ is valid if and only if an embedding obtained from the configuration (1) does not place more than
Figure 4.2: Here we see an assembly, a valid configuration, and an invalid configuration. In the third image, because of the orientations of bonds 1, 2, and 4, bond 3 is between two tiles that are not connected, making the configuration invalid.

one tile at any tile location, (2) doesn’t bond tiles through the same space, and (3) does not have contradicting bond loops. To elaborate on (2), while 4 glues can all be adjacent at one point, we allow them to bind in pairs in “Up” or “Down” orientations but do not allow both pairs to bind across the gap in “Straight” orientations. To elaborate on (3), contradicting bond loops occur when placing a loop of tiles that are all bound in a loop causes the last tile to be placed at a location that is not adjacent to the first tile, therefore making the loop unable to close. Examples of configurations that follow and contradict (3) are given in Figure 4.2. Note that two embeddings that use different initial tiles and initial placements but the same configuration will be equivalent up to rotation and translation.

Let $\alpha$ be an assembly and $c_\alpha$ and $c'_\alpha$ be valid configurations of $\alpha$. If for every flexible bond $b \in \alpha$ either $c_\alpha(b) = Up$ and $c'_\alpha(b) = Down$, $c_\alpha(b) = Down$ and $c'_\alpha(b) = Up$, or $c_\alpha(b) = Straight$ and $c'_\alpha(b) = Straight$, we say that $c_\alpha$ is the chiral configuration of $c'_\alpha$ and vice versa. Note that the embeddings achieved from $c_\alpha$ and $c'_\alpha$ are reflections of each other. We refer to the special reconfiguration of an assembly to its chiral as inversion. We define a pattern of bond orientations, or simply just a pattern, to be a configuration and its chiral.

Given an assembly $\alpha$ and two different embeddings $e_\alpha$ and $e'_\alpha$, we say that $e_\alpha$ and $e'_\alpha$
are equivalent, written \( e_\alpha \equiv e'_\alpha \), if one can be rotated and/or translated into the other. If two embeddings are equivalent, this means they were computed from the same configuration, although possibly using a different placement for the initial tile.

We define the set of all valid configurations of \( \alpha \) as \( \mathcal{C}(\alpha) \). We say that an assembly \( \alpha \) is rigid if (1) \( |\mathcal{C}(\alpha)| = 1 \), or (2) \( |\mathcal{C}(\alpha)| = 2 \) and the two valid configurations are chiral versions of each other. Conversely, if \( \alpha \) is not rigid, we say that it is flexible.

The frontier of a configuration \( c_\alpha \), denoted \( \partial T c_\alpha \), is the set composed of all pairs \((t, B)\) where \( t \in T \) is a tile type from tile set \( T \) and \( B \) is a set of up to 4 tile/glue pairs such that an embedding of \( c_\alpha \) would place each tile adjacent to one location such that a tile of type \( t \) could bind to each glue for a collective strength greater than or equal to the temperature parameter \( \tau \). Given an assembly \( \alpha \) and a set of valid configurations \( \mathcal{C}(\alpha) \), we define the multiset of frontier locations of assembly \( \alpha \) across all valid configurations to be \( \hat{\partial} T \alpha = \bigcup_{c_\alpha \in \mathcal{C}(\alpha)} \partial T c_\alpha \), i.e. \( \hat{\partial} T \alpha \) is multiset resulting from the union of the sets of frontier locations of all valid configurations of \( \alpha \).

Given assembly \( \alpha \) and valid configuration \( c_\alpha \), \#(\alpha) is the maximum number of new bonds which can be formed across adjacent tile edges in an embedding of \( \alpha \) which are not already bound in \( \alpha \) (i.e. these are tile edges which have been put into placements allowing bonding in configuration \( c_\alpha \) but whose bonds are not included in \( \alpha \)). We then define \( \mathcal{C}_{\text{max}}(\alpha) = \{ c_\alpha | c_\alpha \in \mathcal{C}(\alpha) \text{ and } \forall c'_\alpha \in \mathcal{C}(\alpha), \#(c_\alpha) \geq \#(c'_\alpha) \} \). Namely, \( \mathcal{C}_{\text{max}}(\alpha) \) is the set of valid configurations of \( \alpha \) in which the maximum number of bonds can be immediately formed.

Given an assembly \( \alpha \) in FTAM system \( T \), a single step of the assembly process intuitively proceeds by first randomly selecting a frontier location from among all frontier
locations over all valid configurations of $\alpha$. Then, a tile is attached at that location to form a new assembly $\alpha'$. Next, over all valid configurations of $\alpha'$, a configuration is randomly selected in which the maximum number of additional new bonds can be formed (i.e. the addition of the new tile may allow for additional bonds to form in alternate configurations, and a configuration which maximizes these is chosen), and all possible new bonds are formed in that configuration, yielding assembly $\alpha''$. Assuming that $\alpha$ was not terminal and thus $\alpha'' \neq \alpha$, we denote the single-tile addition as $\alpha \rightarrow T \alpha''$. To denote an arbitrary number of assembly steps, we use $\alpha \rightarrow T^* \alpha''$. For an FTAM system $T = (T, \sigma, \tau)$, assembly begins from $\sigma$ and proceeds by adding a single tile at a time until the assembly is terminal (possibly in the limit). (See Algorithms 1 and 2 for pseudocode of the assembly algorithms.) For any $\alpha'$ such that $\sigma \rightarrow T^* \alpha'$, we say that $\alpha'$ is a producible assembly and we denote the set of producible assemblies as $A[T]$. We denote the set of terminal assemblies as $A\square[T]$.

Note that in this section we have provided what is intended to be an intuitively simple version of the FTAM in which the full spectrum of all possible configurations of an assembly are virtually explored at each step, and only those which maximize the number of bonds formed at every step are selected. Logically, this provides a model in which assemblies reconfigure into globally optimal configurations, in terms of bond formation, between each addition of a new tile. Clearly, depending on the size of an assembly and the degrees of freedom of various components afforded by flexible bonds, such optimal reconfiguration could conceivably be precluded by faster rates of tile attachments. Various parameters which seek to balance the amount of configuration-space exploration versus tile attachment rates have been developed to study more kinetically realistic dynamics, but are beyond the scope of this thesis.
Algorithm 1 A procedure to perform one step of the self-assembly process of FTAM system $T$

1: procedure ASSEMBLY-STEP($\alpha$, $T$) \Comment{takes an assembly $\alpha$ and FTAM system $T$}
2: \hspace{1em} if $|\partial^T \alpha| = 0$ then
3: \hspace{2em} return $\alpha$ \Comment{no frontier locations remain, $\alpha$ is terminal}
4: \hspace{1em} else
5: \hspace{2em} Uniformly at random select $(t, B) \in \hat{\partial T} \alpha$ \Comment{select a frontier location}
6: \hspace{2em} Attach a tile of type $t$ with bonds to tiles in $B$, $\alpha \rightarrow^T_T \alpha'$ \Comment{add a tile}
7: \hspace{2em} Uniformly at random select $c'_\alpha \in C_{max}(\alpha')$ \Comment{find new-bond-maximizing configuration}
8: \hspace{2em} Form all bonds possible in $c'_\alpha$ to yield $\alpha''$ \Comment{form those bonds}
9: \hspace{2em} return $\alpha''$ \Comment{return the new assembly}
10: end if
11: end procedure

Algorithm 2 A procedure to perform the self-assembly process of FTAM system $T$

1: procedure FULL-ASSEMBLY($\alpha$, $T$) \Comment{takes an assembly $\alpha$ and FTAM system $T$}
2: \hspace{1em} $\alpha' = \text{ASSEMBLY-STEP} (\alpha, T)$
3: \hspace{1em} if $\alpha =\alpha'$ then
4: \hspace{2em} return $\alpha'$
5: \hspace{1em} else
6: \hspace{2em} return FULL-ASSEMBLY($\alpha'$, $T$)
7: \hspace{1em} end if
8: end procedure

4.2 Controlling flexibility to build structures

Our goal in this section is to deterministically assemble certain shapes in the FTAM at temperature two. This goal was explored to an extent in [38]. However, we will extend the analysis done in [38] by giving a more thorough investigation of vertices and assembly processes. To begin, we define a \textit{shape} to be a collection of connected tile locations. A shape is invariant through translation and rotation. Rather than go through an endless case-by-case analysis of all possible shapes, we focus on collections of 2D tile locations that form the outlines of three-dimensional shapes. We refer to these 3D shapes as \textit{polycubes} and the
sets of 2D tile locations on their outer surfaces as *outlines*. We say that an FTAM system $\mathcal{T} = (T, \sigma, \tau)$ *deterministically assembles* a shape $s$ if the embedding of all configurations $C_\alpha$ of all terminal assemblies $\mathcal{A}_\subseteq[T]$ of the system $T$ have shape $s$.

Due to the definition of the model, the most prominent additional challenge that is present in FTAM systems over traditional 2D aTAM systems is controlling the orientation of different faces in the assembly relative to one another as the assembly process is occurring. In which case, the approach that we use to demonstrate shape building in the FTAM is to make an *edge frame* for each polycube using unique tile types and filling in each face. We define an edge frame to be the collection of the outer-most tiles of each face in the outline of a polycube. For now, we will make the assumption that every edge of the shape is connected and will address this later in the section. We claim that studying edge frames is sufficient for unveiling the power of the FTAM to orient new faces in the assembly process since, intuitively, the cooperation of other tiles on the edges of adjacent faces doesn’t provide additional help in correctly orienting those faces over just the tiles at the vertex. This intuition stems from the idea that the faces of a shape incident on a vertex interact on the same axes that the individual tiles incident on a vertex do. Once an edge frame has been built, it can be filled out by utilizing “filler tiles”. On perfectly square faces, this can trivially be done, with filler tiles allowing to attach as the assembly grows. However, in cases where the face has a concave corner, a rectangular decomposition of the face with each rectangle being assigned a unique filler tile would prevent the filler tiles from overgrowing their bounds.

The first obstacle in building shapes in the FTAM is the inherent property of FTAM assemblies to always be able to reconfigure into a chiral configuration. As mentioned in Section 4.1, this is done by simply flipping all the “up” orientations to “down” and all the
“down” orientations to “up” in the configuration. Due to our definition of deterministic assembly, it is therefore impossible for the FTAM to build any shape unless it is symmetric. Symmetry mitigates this issue since any configuration of a symmetric shape and its chiral yield the same shape since the embeddings are reflections of each other. However, this makes symmetry our first restriction for shapes that can be built in the FTAM.

Another big factor in deciding whether a polycube outline can be deterministically assembled is the types of vertices that compose the polycube. Therefore, we continue our analysis by breaking down all the possible types of vertices that can exist on a polycube. These can be enumerated by enumerating all polycubes that can fit inside a $2 \times 2 \times 2$ space that are distinct up to rotation and reflection. You can see the outcome of this enumeration in Figure 4.3. In each polycube, the vertex type is illustrated at the center point of the $2 \times 2 \times 2$ space. The illustration has labels to later reference each vertex type.

![Figure 4.3: All possible polycubes that can fit inside of a $2 \times 2 \times 2$ space, and furthermore, all possible vertex types that could exist on a polycube.](image)

We categorize the types of vertices into two groups, *simple* and *complex*. In the enumeration in Figure 4.3, the polycubes in the blue squares actually don’t have a vertex in the center. The vertices in the polycubes in red (1,2,5,9) have three edges and three faces.
incident on the center point, creating what we refer to as a simple vertex. Of these 4 vertices, 1 and 9 are the same vertex type, which we will refer to as a convex vertex, and 2 and 5 are the same vertex type, which we will refer to as a concave vertex. The simple vertices can also be visualized in Figure 4.4. The vertices in the polycubes in brown (3,4,6,7) have more than three edges and more than three faces incident on the center point, creating what we refer to as a complex vertex. All of these complex vertices are unique, and we will refer to them by their number. The polycube in yellow (8) is a special case in which there are more than three edges and more than three faces incident on the center point, but the polycube is arranged in a way that the center point can be thought of as two different simple convex vertices, one for each location that is missing a cube.

In addition to the vertex type, the system must also be able to deterministically assemble the vertex from the correct perspective. A perspective is the relative direction that the new edges of a vertex are pointing with respect to the tiles of the original edge. A vertex can be symmetric, meaning all edges have the same perspective, semi-symmetric, meaning some edges have the same perspective, or asymmetric, meaning no edges have the same perspective. Semi-symmetric and asymmetric vertices have to differentiate between the different perspectives the vertex can exist in. The simple convex vertex and vertex 3 are both symmetric, meaning they have only one perspective each. Vertex 4 and 7 are semi-symmetric, with vertex 4 having 2 perspectives (even though it has 4 edges) and vertex 7 having 3 perspectives (even though it has 6 edges). The simple concave vertex and vertex 6 are asymmetric, with the simple concave vertex having 3 perspectives (and 3 edges) and the vertex 6 having 5 perspectives (and 5 edges). An example of different perspectives can be seen in the difference between Figure 4.4c and Figure 4.4d. Since these are the same
vertex but different perspectives, we have elected to refer to them both as concave vertices, in contrast to [38] where they are differentiated into concave and “combined” vertices. All together, there are 15 different perspectives.

![Image](image.png)

**Figure 4.4:** Illustration of (a) an original edge, (b) a convex vertex, (c) a concave vertex, from one unique perspective, and (d) a concave vertex, from another perspective.

When building vertices in the assembly process, we have control of two properties when designing our system that can allow us to control the relative orientation of the new edges and faces (under the assumption that we are building these vertices using a loop of tiles with unique glues between them). These properties are the loop length and the bond sequence. The *loop length* is the number of tiles that we attach in a loop (only counting those incident on the vertex). The *bond sequence* is the ordering of rigid and flexible bonds within that loop. These loops for the complex vertices can be visualized in Figure 4.5. We call the combination of a loop length and bond sequence a *tiling protocol*. If a perspective has a loop length and bond sequence that is unique among the set of all 15 perspectives, then utilizing that tiling protocol can only result in that vertex and perspective. However, if multiple perspectives share a loop length and bond sequence, then utilizing that tiling protocol can non-deterministically choose between them (unless some external factor is also restricting this) and switch at each step in the assembly process.
We will now look at the loop lengths and bond sequences of different vertices. For bond sequences, we will use the notation \((b, b, ..., b)\) where \(b \in \{R, F\}\) and \(R\) stands for rigid and \(F\) stands for flexible. The first bond in the sequence will represent the edge assembling up to the vertex in the assembly process, and will therefore always be flexible. The other bonds in the sequence will represent the other bonds within the loop of tiles incident on the vertex, following the ordering set by the loop in either direction, without loss of generality. The set of all bond sequences for a single vertex is a non-repeating cyclic permutation group, minus the elements that begin with a rigid bond instead of a flexible bond.

We start with the simple vertices. First is the simple convex vertex. It has a loop length of 3 tiles and a bond sequence \((F, F, F)\). It is symmetric, meaning we don’t have to differentiate between the edges. Therefore, to make a simple convex vertex, an edge will just initiate a protocol where the last two tiles of the edge end in flexible glues and another tile that matches both glues attaches to complete the loop. The next vertex is the simple concave vertex. It has a loop length of 5 and can have a bond sequence of either \((F, R, R, F, F)\), \((F, F, R, R, F)\), or \((F, F, F, R, R)\). Using each bond orientation will result in a different perspective, meaning all 3 perspectives can be deterministically assembled. Therefore, attaching the loop of tiles with the first bond sequence will yield the vertex in Figure 4.4d, the second bond sequence will yield the vertex in Figure 4.4c, and the third...
sequence will yield the mirror opposite of the vertex in Figure 4.4d.


Vertex 3 and 7, however, share a combination of a loop length and bond sequence. Both have a loop length of 6 and a bond sequence of \((F, F, F, F, F, F)\). Because of this, attaching a loop of 6 tiles using all flexible bonds at the end of an edge can result in either vertex. In addition, since flexible bonds can “mimic” rigid bonds using a “Straight” orientation, the loop of tiles can even configure into vertex 4. Note that the reverse is mitigated by the fact vertex 4 has rigid bonds and no bonds in vertex 3 or 7 are “Straight”. All together, it can end up in the one perspective from vertex 3, one of the two perspectives from vertex 4, or one of the three perspective from vertex 7. Given all these possibilities, vertex 3 and 7 cannot be deterministically assembled.

**Assembly Process.** Now, we consider the assembly process. Let’s assume we start with a seed that is just the three tiles in a simple convex vertex. Notice that as the assembly process starts, the seed vertex and the edges that are growing out from it can invert as a whole but cannot otherwise reconfigure (since that would require removing a bond from the
assembly). For assembling an edge, we outline a trivial protocol. Side 1 and side 2 refer to the two columns of tiles on the two faces that make up the edge. The hinge refers to the series of flexible bonds between tiles on side 1 and tiles on side 2.

1. An exposed rigid double strength glue on side 1 of the edge will attach a new tile \( t \) on side 1,

2. A flexible glue on tile \( t \) on side 1 and an exposed rigid glue on side 2 will cooperate to attach a new tile \( t' \) on side 2 of the edge,

3. A rigid double strength glue on tile \( t' \) on side 2 of the edge will attach a new tile \( t'' \) on side 2, and

4. A flexible glue on tile \( t'' \) on side 2 and a rigid glue on tile \( t \) on side 1 will cooperate to attach a new tile \( t''' \) on side 1 of the edge.

An edge can grow indefinitely by repeating this process using unique glues to grow up to a certain length. Notice that each new tile attaches using at least one rigid bond, meaning that, additional flexibility cannot be added to the edge past the flexibility of the hinge. Furthermore, there will only ever be one frontier location (per configuration, if multiple, but these are the same tiles and bonds) on the edge at any assembly step, leaving no room for non-determinism. Using this protocol, the assembly can grow up to a vertex, where it can attach the loop of tiles that make this new vertex. As long as the new vertex is not a reconfigurable vertex, it will be forced to take a configuration that agrees with configuration of the seed vertex. By this, we mean that, if the seed vertex were to invert at this point, the edge connecting the two vertices would invert, and the new vertex would therefore be
forced to invert. This cause-effect relationship is true for any vertices (again excluding the reconfigurable vertices) connected by an edge, which means that, if any bond in the partial assembly were to reorient, the whole partial assembly must invert, i.e. inversion is the only possible reconfiguration. An example of the assembly of an edge frame starting from a potential seed is shown in Figure 4.6.

Figure 4.6: An assembling edge frame starting from a potential seed. Each edge grows up to a vertex and into other edges until the whole frame has filled out.

We now prove a claim that assembling in the correct configuration or the chiral configuration is identical (since both configurations have the same frontier) and will therefore yield the same shape.

Claim 4.2.1. Every frontier location \( f \) in an assembly \( \alpha \) for a given configuration \( c_\alpha \) has a corresponding frontier location \( f' \) in \( \alpha \) in the chiral configuration \( c'_\alpha \), such that attaching \( f \) to \( \alpha \) in \( c_\alpha \) produces the same assembly but in the chiral configuration of attaching \( f' \) to \( \alpha \) in \( c'_\alpha \).

Proof. Notice that a frontier location in the FTAM is dependent on 12 neighboring tile locations, an “Up”, “Straight”, and “Down” location for each of the 4 sides of the tile. Also remember chiral configurations of an assembly \( \alpha \) produce embeddings of \( \alpha \) that are the reflections of each other. Now, take any frontier location \( f \) in \( c_\alpha \). By reflecting an embedding
of \( c_\alpha \) over the plane that \( f \) exists in, the 12 tile locations that make \( f \) into a frontier location will still be neighboring \( f \), with the “Up” and “Down” neighboring locations switching places and also reflecting, thereby keeping the same glues incident on the location of \( f \). Since all the same glues are incident on the tile location, this location, which we will call \( f' \), is also a frontier location in \( c'_\alpha \) with the same tile type as in \( c_\alpha \), even if \( c'_\alpha \) includes some translation or rotation. Since the frontier locations are on the plane of symmetry that we used to get the chiral configurations, adding the tile to the assembly in either configuration will produce two configurations that are also chiral configurations of each other.

Once the assembly process has finished, the terminal assembly could also flip between the correct shape in its chiral. As previously mentioned, when there is at least one plane of symmetry in the shape, then reconfiguration in the assembly process actually will not prevent the system from being deterministic. This is because the chiral of a symmetric polycube is itself. Therefore, although the system will technically make two different terminal assemblies, one can be rotated into the other, meaning that the two different terminal assemblies have the same shape by definition, making the system deterministic.

Multiple Edge Frames. Up to this point, we have assumed all the edges in a polycube are connected. However, this is not always the case. For example, anytime two pieces of a shape are connected by a set of coplanar tiles (i.e. when the face graph has a cut vertex). This is similar to the idea of “control modules” in [38]. Shapes like this are a problem because they require multiple edge frames to build, and similar to the chirality of asymmetric shapes, additional edge frames can also have chiral reconfigurations. Therefore, disagreeing chiralities of the edge frames can configure the terminal assembly of a system.
into a shape that is neither the intended shape nor its chiral. In general, each additional edge frame doubles the number of configurations that the terminal assembly can exist in, only one of which (or two, if symmetric) is the desired shape. There are some exceptions to this exponential growth, such as blocking and symmetry.

Blocking refers to when the faces surrounded by one edge frame would collide with the faces of another if the chiralities of the edge frames disagreed. This is the case in the example given in Figure 4.7. In these situations, even if the additional edge frames are configured to the wrong chirality during the assembly process, eventually the tiles with the potential to collide will be added to the assembly and force that configuration to be invalid. In Figure 4.7, you can see how the inversion of the blue and green edge frames at the same time would cause the collision of tiles in the middle of the assembly. This doesn’t ensure a correct configuration, since one can invert while the other remains, but it does rule out the possibility of both the blue and green edge frames reconfiguring together.

One other aspect of a shape with multiple edge frames that may reduce the number of possible configurations it can exist in is, like with full shapes, symmetry of the edge frames. To utilize the same example, notice that, in the shape in Figure 4.7, the inversion of the blue edge frame and not the green, and the inversion of the green edge frame and not the blue, yield two configurations that can be rotated into each other. This is a result of there being a plane of symmetry that exists through all three edge frames. Therefore, these configurations that are technically inversions of each other can actually be rotated into one another instead of having to be reflected across a plane.

While two additional edge frames will traditionally yield 4 different configurations, given that one configuration has a blocking conflict (both the blue and green edge frames
Figure 4.7: Example of a shape with multiple edge frames. The left shows the shape assembled with tiles. The right shows the outline of the shape and how it is broken into 3 disconnected pieces.

inverted simultaneously) and two of the other configurations are actually the same shape (the blue edge frame inverted and not the green and the green edge frame inverted and not the blue), this assembly actually only has 2 different shapes it can exist in. These two shapes being the correct shape with no inversion and the shape resulting from inverting exactly one of the blue or green edge frame is inverted and not the other.

Summary. Combining the results of this section, we get the following theorem.

Theorem 2. A temperature two FTAM system can deterministically assemble the outline of any polycube that meets the following conditions:

1. the polycube is symmetric,

2. there are no reconfigurable vertices in the polycube, and

3. the edges of the polycube are all connected
Note that, through the discussion of the caveats above, there are additional shapes that can also be built by FTAM systems. Keeping in the subspace of polycube outlines, we already mentioned how issues with reconfigurable vertices can be rectified with multiple edges cooperating to force orientation and how issues with multiple edge frames can be rectified with symmetry and blocking. However, these cases (and more) require more case-by-case based analysis, rather than being studied from a general perspective.
5 The Effects of Modeling Diffusion on Simulation

The last model that we examine in relation to the abstract Tile Assembly Model is the
Spatial abstract Tile Assembly Model (Spatial aTAM). We introduced this model in [17]. It is
a 3D version of the standard aTAM, with the addition of a diffusion constraint. Colloquially,
this new property restricts tiles from attaching to the assembly in locations that don’t have
an unobstructed diffusion path from infinitely far away. The study of this model is important
since we have results involving the computational ability of other self-assembly models, but
these models don’t take into account this realistic property of physical systems. Therefore,
in applying this constraint we are making our models more “realistic”, thereby narrowing
the gap between our theoretical results and experimental possibilities.

The main result that we will look at is the property of intrinsic universality (IU),
or the ability of a computational model to simulate itself. As previously mentioned in the
introduction, the Turing machine model is IU, meaning there is one Turing machine (i.e.
one set of states and transitions) that, by taking the encoded definition of another Turing
machine as input, can simulate any other Turing machine and produce the same result. In
our case, within an IU tile assembly model, there exists a tile set that, given a seed that
encodes any arbitrary system in the model, can simulate that system given some scale factor
and produce a scale-up version of one of the system’s producible terminal assemblies. In [17],
we show that the traditional 3D aTAM is IU, in addition to actually designing out a universal
tile set and implementing it in a simulator. In this chapter, we look at the same property for
the model with the diffusion constraint. In determining this property for the Spatial aTAM,
we gain insight on the ability of spatial systems to simulate the diffusion constraint on a macroscale.

5.1 Formal definition of the Spatial abstract Tile Assembly Model

In this section, we will define the terminology surrounding the Spatial aTAM that will allow us to prove results about it. Given that the Spatial aTAM is a variation of the 3D aTAM, we refer the reader to Section 2.2 for the basics of tiles types, tiles, assemblies, systems, seeds, temperature, frontier, assembly sequences, and terminal assemblies. Here, we will define the diffusion constraint, constrained subspaces, constraining assemblies, equivalent production, equivalent dynamics, simulation, and finally, intrinsic universality.

5.1.1 Diffusion constraint

As previously stated, a Spatial aTAM system is one where, in addition to all of the normal requirements for tile attachment, a tile can only attach to an assembly if there exists a contiguous path from the node representing the attachment location to a node outside of the minimal bounding box of the assembly in the graph corresponding to the lattice $\mathbb{Z}^3$, such that none of the points along the path are in the domain of the assembly. We restrict this path such that adjacent nodes in the path must be 1 unit away from each other, i.e. adjacent nodes must be $(\pm x, \pm y, \pm z)$ in relation to each other and not diagonal. We call such a path a diffusion path.

Notice that, since tiles never detach in the aTAM, once a given location has had all diffusion paths blocked, i.e. it is surrounded by the assembly, no tile will ever be able to attach in that location. We say that the spatial constraint has been invoked on such a tile

67
location. We call a connected set of locations in which tiles cannot attach due to the spatial constraint a \textit{constrained subspace}. The set of all tiles that are adjacent to a constrained subspace is called the \textit{constraining subassembly}. Notice that a constraining subassembly is not actually a connected assembly, as it will always contain disconnected sets of tiles (due to the diffusion path only including $\pm x$, $\pm y$, and $\pm z$ movements). In other words, the constraining subassembly is the set of all tiles such that, if any single tile were removed, the constrained subspace would either no longer be constrained or would now contain the location of the removed tile.

5.1.2 Simulation

Since we are looking at a property that involves simulation, we must formally define what it means for one tile assembly system to "simulate" another. Our definitions come from [32] with the natural modifications to extend from 2D to 3D. Intuitively, simulation of a system $T$ by another system $S$ is done by utilizing some scale factor $m \in \mathbb{Z}^+$ such that $m \times m \times m$ cubes of tiles in $S$ represent individual tiles in $T$, and there is a "representation function" which is able to interpret the assemblies of $S$ as assemblies in $T$.

From this point on, let $T$ be a tile set and let $m \in \mathbb{Z}^+$. An \textit{$m$-block macrotile} over $T$ is a partial function $\alpha : \mathbb{Z}_m^3 \rightarrow T$, where $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$. Let $B_m^T$ be the set of all $m$-block macrotiles over $T$. The $m$-block with no domain is said to be \textit{empty}. For a general assembly $\alpha : \mathbb{Z}^3 \rightarrow T$ and $(x', y', z') \in \mathbb{Z}^3$, define $\alpha_{(x', y', z')}^m$ to be the $m$-block macrotile defined by $\alpha_{(x', y', z')}^m(i_x, i_y, i_z) = \alpha(mx' + i_x, my' + i_y, mz' + i_z)$ for $0 \leq i_x, i_y, i_z < m$. For some tile set $S$, a partial function $R : B_m^S \rightarrow T$ is said to be a \textit{valid $m$-block macrotile representation} from $S$ to $T$ if for any $\alpha, \beta \in B_m^S$ such that $\alpha \subseteq \beta$ and $\alpha \in \text{dom } R$, then
\( R(\alpha) = R(\beta). \)

For a given valid \( m \)-block macrotile representation function \( R \) from tile set \( S \) to tile set \( T \), define the assembly representation function\(^1\) \( R^* : \mathcal{A}^S \rightarrow \mathcal{A}^T \) such that \( R^*(\alpha') = \alpha \) if and only if \( \alpha(x,y,z) = R\left(\alpha_{(x,y,z)}^m\right) \) for all \( (x,y,z) \in \mathbb{Z}^3 \). For an assembly \( \alpha' \in \mathcal{A}^S \) such that \( R^*(\alpha') = \alpha \), \( \alpha' \) is said to map cleanly to \( \alpha \in \mathcal{A}^T \) under \( R^* \) if for all non-empty blocks \( \alpha_{(x,y,z)}^m \), \( (x,y,z) + (u_x,u_y,u_z) \in \text{dom}(\alpha) \) for some \( (u_x,u_y,u_z) \in U_3^m \) such that \( u_x^2 + u_y^2 + u_z^2 \leq 1 \), or if \( \alpha' \) has at most one non-empty \( m \)-block \( \alpha_{(x,y,z)}^m \). In other words, \( \alpha' \) may have tiles on macrotile blocks representing empty space in \( \alpha \), but only if that position is adjacent to a tile in \( \alpha \). We call such growth “around the edges” of \( \alpha' \) fuzz and thus restrict it to be adjacent to only valid macrotiles, but not diagonally adjacent (i.e. we do not permit diagonal fuzz).

In the following definitions, let \( T = (T, \sigma_T, \tau_T) \) be a TAS, let \( S = (S, \sigma_S, \tau_S) \) be a TAS, and let \( R : B^S_m \rightarrow T \).

**Definition 1.** We say that \( S \) and \( T \) have equivalent productions (under \( R \)), and we write \( S \Leftrightarrow T \) if the following conditions hold:

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

**Definition 2.** We say that \( T \) follows \( S \) (under \( R \)), and we write \( T \vdash^R S \) if \( \alpha' \rightarrow^S \beta' \), for some \( \alpha', \beta' \in \mathcal{A}[S] \), implies that \( R^*(\alpha') \rightarrow^T R^*(\beta') \).

The next definition specifies that every time \( S \) simulates an assembly \( \alpha \in \mathcal{A}[T] \), there must be at least one valid growth path in \( S \) for each of the possible next steps that \( T \) could follow. 

\(^1\)Note that \( R^* \) is a total function since every assembly of \( S \) represents some assembly of \( T \); the functions \( R \) and \( \alpha \) are partial to allow undefined points to represent empty space.
make from $\alpha$ which results in an assembly in $\mathcal{S}$ that maps to that next step.

**Definition 3.** We say that $\mathcal{S}$ models $\mathcal{T}$ (under $R$), and we write $\mathcal{S} \models_R \mathcal{T}$, if for every $\alpha \in \mathcal{A}[\mathcal{T}]$, there exists $\Pi \subset \mathcal{A}[\mathcal{S}]$ where $R^*(\alpha') = \alpha$ for all $\alpha' \in \Pi$, such that, for every $\beta \in \mathcal{A}[\mathcal{T}]$ where $\alpha \xrightarrow{T} \beta$, (1) for every $\alpha' \in \Pi$ there exists $\beta' \in \mathcal{A}[\mathcal{S}]$ where $R^*(\beta') = \beta$ and $\alpha' \xrightarrow{S} \beta'$, and (2) for every $\alpha'' \in \mathcal{A}[\mathcal{S}]$ where $\alpha'' \xrightarrow{S} \beta'$, $\beta' \in \mathcal{A}[\mathcal{S}]$, $R^*(\alpha'') = \alpha$, and $R^*(\beta') = \beta$, there exists $\alpha' \in \Pi$ such that $\alpha' \xrightarrow{S} \alpha''$.

**Definition 4.** We say that $\mathcal{S}$ simulates $\mathcal{T}$ (under $R$) if $\mathcal{S} \equiv_R \mathcal{T}$ (equivalent productions), $\mathcal{T} \models_R \mathcal{S}$ and $\mathcal{S} \models_R \mathcal{T}$ (equivalent dynamics).

### 5.1.3 Intrinsic universality

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

Let $\text{REPR}$ denote the set of all macrotile representation functions (i.e., $m$-block macrotile representation functions for some $m \in \mathbb{Z}^+$). Define $\mathcal{C}$ to be a class of tile assembly systems, and let $U$ be a tile set. Note that each element of $\mathcal{C}$, $\text{REPR}$, and $\mathcal{A}_{U<\infty}$ is a finite object, hence encoding and decoding of simulated and simulator assemblies can be defined to be computable via standard models such as Turing machines and Boolean circuits.

**Definition 5.** We say $U$ is **intrinsically universal** for $\mathcal{C}$ at temperature $\tau' \in \mathbb{Z}^+$ if there are computable functions $\mathcal{R} : \mathcal{C} \to \text{REPR}$ and $\mathcal{S} : \mathcal{C} \to \mathcal{A}_{U<\infty}$ such that, for each $\mathcal{T} = (T, \sigma, \tau) \in \mathcal{C}$, there is a constant $m \in \mathbb{N}$ such that, letting $R = \mathcal{R}(\mathcal{T})$, $\sigma_T = \mathcal{S}(\mathcal{T})$, and $U_T = (U, \sigma_T, \tau')$,
\( \mathcal{U}_T \) simulates \( \mathcal{T} \) at scale \( m \) and using macrotile representation function \( R \).

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

**Definition 6.** We say that \( \mathcal{U} \) is intrinsically universal for \( \mathcal{C} \) if it is intrinsically universal for \( \mathcal{C} \) at some temperature \( \tau' \in \mathbb{Z}^+ \).

**Definition 7.** We say that \( \mathcal{C} \) is intrinsically universal if there exists some \( \mathcal{U} \) such that \( \mathcal{U} \) is intrinsically universal for \( \mathcal{C} \).

### 5.2 Intrinsic universality in the Spatial abstract Tile Assembly Model

**Theorem 3.** The Spatial abstract Tile Assembly Model is intrinsically universal.

To prove Theorem 3, we will provide an augmentation of the construction [17] used to prove that the traditional 3D aTAM is intrinsically universal. Before we get into the augmentations that we make, we will give a brief overview of the original construction. Note that this original construction is complex and requires many details to ensure that it works, the majority of which we exclude here. We refer the reader to [17] for more in-depth explanation about how the construction works, as well as fully detailed proofs of the construction’s correctness. This system can be broken down into four modules or functional subassemblies: the **Genome**, the **Adder Array**, the **Bracket**, and the **External Communication**. There will be an instance of each of these modules in every macrotile \( L \) in the simulator system \( \mathcal{U}_T \) that resolves to represent a tile in location \( l \) in the simulated system \( \mathcal{T} \). We provide brief descriptions of the functions of each here.
• **Genome**: This module contains an encoding of the system to be simulated, $\mathcal{T}$, in the form of a look-up table that takes as input the tile types of neighboring macrotiles and outputs every potential tile type that could form a bond with every combination of neighboring macrotiles along with the strength of the corresponding bond. The genome also contains instructions to build the other modules listed here.

• **Adder Array**: This module is responsible for determining, for each tile type $t \in \mathcal{T}$, if there are enough glues incident on the current macrotile for it to begin to represent a tile of type $t$ under $R$. It does this by adding up the bond strengths (from the genome) with which a tile of type $t$ could attach in the current location and making sure the total is sufficient for attachment. This module is quite literally an array of 63 adder gadgets which are commonly used in tile constructions, one for each possible subset of the 6 incoming neighbors excluding the empty set.

• **Bracket**: Once the adder array determines the tile types into which the macrotile can resolve, this module picks one tile type non-deterministically (if there is a choice). If the macrotile can resolve into multiple tile types, the adder will provide an outgoing signal corresponding to each one. These signals are then forced to move through an actual bracket. Each signal will potentially race another to a location in which they can both place a tile. The first signal to place a tile in that location proceeds to the next layer of the bracket until only one is left.

• **External Communication**: This module carries an encoding of the decided upon tile type (as output from the bracket) from the current macrotile to all neighboring macrotiles locations.
Now, we will describe the process by which one macrotile $L$ goes from empty space to fully grown. We refer to the transition of a macrotile from mapping to empty space in $T$ to mapping to a tile in $T$ under some representation function $R^*$ as differentiation.

1. Once a neighboring macrotile location has differentiated, it exports a copy of the genome and its external communication to macrotile $L$.

2. The genome propagates around $L$ and initiates growth of the other three modules.

3. The incoming external communication modules from differentiated neighbors grow into the genome to query for whether or not their glues could contribute to the differentiation of $L$. It’s important to note that the arrival of these signals is nondeterministic and that the construction is robust to the arbitrary arrival order of these signals.

4. The information from the previous step is sent to the adder array to determine if enough glues are present to allow simulation of the attachment of specific tile types.

5. Encodings of potential tile types enter the bracket and “race” each other to the end where one non-deterministically wins and becomes the representative tile type.

6. The winning tile type leaves the bracket and grows to the external communication.

7. The genome and external communication modules are propagated to the neighboring macrotiles locations to initiate subsequent growth.

A visualization of this process can be seen in Figure 5.1 and an illustration of the propagation of the genome can be seen in Figure 5.2. The genome is encoded along three nested bands which all contain identical information. Each band is capable of growing the...
other two and the purpose of the bands is to easily allow the propagation of the genome information to neighboring macrotiles in each direction. Additionally, all information that is passed between different modules in the same macrotile or between neighboring macrotiles during external communication is transmitted using gadgets we call datapaths. A *datapath* consist of rows of tiles which contain, encoded in their forward facing glues, a combination of data and instructions for propagating. Using standard machinery in tile assembly, these datapaths propagate according to their instructions, which can include dynamics such as moving forward some fixed distance using counters, rotating the data so that propagation continues in a different direction, and waiting for a signal before continuing propagation.

Essentially, a macrotile block $L$ in the terminal assembly of $\mathcal{U}_\mathcal{T}$ (representing a tile location $l$ in the terminal assembly of $\mathcal{T}$) can be in one of three states. If $l$ is not adjacent to any tile in the terminal assembly of $\mathcal{T}$, $L$ will be completely empty in the terminal assembly of $\mathcal{U}_\mathcal{T}$. If $l$ is adjacent to a tile but does not have enough incident glues to be a frontier

---

**Figure 5.1:** Illustration of the steps in which a macrotile goes from mapping to empty space to differentiating into representing a specific tile type in the simulated system and propagating that information to its neighbors.
location, \( L \) will have all four modules set up but no tile types will be output from the adder array to the bracket. Finally, if \( l \) represents a tile, \( L \) will have all four modules set up and an encoding of that tile type will have left the adder array, made it through the bracket, and be outputted to neighboring macrotile locations by the external communication.

In addition to this growth paradigm, the construction includes functions \( R \) and \( S \). \( R \) works by using the scale factor of the simulation to determine where the output of the bracket will be in each macrotile block and setting this to be the input of the representation function \( R \). Once this set of relative tile positions within each block is filled, \( R \) reads the encoding within the individual tiles and outputs the corresponding tile type from \( T \). To obtain the seed \( \sigma_T \) of the simulator \( U_T \), the function \( S \) reads the simulated system \( T \) and encodes it into a corresponding genome. This genome is placed in the macrotile locations that map to tile locations filled by the seed in \( T \). Additionally, a hard-coded bracket output is included in each seed macrotile to ensure that the seed of \( U_T \) maps under \( R^* \) to the seed of \( T \). Once the simulation begins, this seed is able to start propagating the genome and external communication modules to neighboring macrotile locations of the seed to start the process of differentiation for those locations and all further growth.
5.2.1 Our augmented construction

The problem with using the original construction for proving intrinsic universality in the diffusion restricted model is that the system is able to grow and differentiate new macrotiles within locations that map to constrained subspaces (i.e. subspaces which are completely sealed off by the tiles of the assembly). To prevent this, we supplement the original construction to use a blocking protocol that will force tiles to attach around the boundary of a macrotile which hasn’t yet differentiated, but still allow diffusion through a series of one-tile-wide pipes until differentiation happens.

The centerpiece of this augmented construction is a module that we’ll subsequently refer to as the Pipe Intersection. Shown in Figure 5.3, this structure helps scale the spatial constraint by tying all of the paths that connect through it to the six faces of the macrotile. Therefore, by adding the central tile location as an input to the representation function $R$, we can have the macrotile differentiate in the exact same assembly step that the diffusion paths between all six neighbors are cut off. In other words, placing a tile in the middle of the pipe intersection (1) blocks any diffusion between the neighboring macrotiles through the current macrotile and (2) causes the current macrotile to differentiate. To implement this new blocking protocol, instead of performing step 7 in the growth sequence of the original construction, we now perform the following sequence of steps after step 6:

1. The genome and external communication modules grow to the boundaries of macrotile location and pause.

2. Pipes are seeded at the pipe intersection and grow to all boundaries (except the top).

3. Boundaries are tiled, starting from the bottom boundary and growing in a spiral around
side boundaries, growing around the I/O datapaths and the ends of the pipes.

4. The pipe intersection is filled from above and the macrotile officially differentiates.

5. Tiles diffuse into the pipes that connect to unconstrained space, attaching until they reach the boundary where they activate the genome and external communication modules to continue growing into neighboring macrotiles.

Figure 5.3: The pipe intersection. This module is important in the simulation of the spatial constraint because it allows multiple paths to be cut off simultaneously when the macrotile differentiates.

To make the blocking protocol we use more concrete, we will present an example of how it is implemented. Note that the construction could be more symmetric and timing efficient at the expense of using more components and having more timing dependencies.

Our implementation of the blocking protocol also makes use of datapaths for passing signals and information and make use of cooperation to enforce sequential processes such that one component must wait on another to complete before processing itself.

For use in discussing our augmentations to the original IU construction, let \( \text{boundary}_{dir} \) be the set of tile locations that form the planar segment that is furthest in the \( dir \) direction for a given macrotile where \( dir \in \{U, N, E, S, W, D\} \). Let the tile location in the center of the pipe intersection be \( \text{loc}_{center} \), since this is the centerpiece of the construction. Let \( side \)
designate the subset of directions \( \{N, E, S, W\} \). Let \( \text{pipe}_{dir} \) be the pipe subassembly that grows from the \( \text{loc}_{center} \) to \( \text{boundary}_{dir} \) for \( dir \in \{N, E, S, W, D\} \).

### 5.2.2 Incoming genome and external communication

To have the correct set up for our blocking protocol to work, we first need to make a few adjustments to how the genome and external communication propagate to new macrotiles. The genome is a good module to make this adjustment in, since an incoming genome signifies that a neighbor has already differentiated. The adjustment is threefold. First, we need the genome and external communication to come in at a new location. Figure 5.7 shows an example of this for the side faces. Next, we need the genome and external communication to exhibit a series of special glues at the boundary when it first comes into the current macrotile. These glues must be exposed in all directions perpendicular to the direction it is traveling. These glues will be used to direct the tiling of the boundary around the datapaths at a later stage.

Most importantly though, we need the incoming genome to generate what we subsequently refer to as an interception gadget (shown in Figure 5.4). This gadget is used later to “cut” a hole in some of the piping to redirect tiles to the “up” direction. Once the pipes start forming, the gadget works by using special glues to cooperate with the attaching tiles such that the pipe can grow past it but will not fill in one location in the direction of the genome. Then, once the macrotile differentiates at a later stage, if there is a diffusion path from the “up” direction, tiles will be able to grow through this hole, along the genome, and signal that the macrotile is clear to propagate its information in the “up” direction. Note there will be 5 inception gadgets, one for each direction other than “up”, which will refer to
Figure 5.4: The interception gadget works by growing out from an incoming genome path into the path of the future pipe that will extend in the same direction. Once the pipe grows, the original sequence of tile placements is blocked, and it instead cooperates with the interception gadget to continue growing but with a hole in the side toward the original genome path. Then, whenever the macrotile differentiates and tiles start growing through the piping, a signal can grow out of the hole and along the genome to activate its (and the corresponding external communication datapath’s) propagation into the neighboring macrotile in the “up” direction. The path of this signal is shown in yellow.

as intercept\(_{\text{dir}}\) where \(\text{dir} \in \{N, E, S, W, D\}\).

5.2.3 Outgoing genome and external communication

The next step in our implementation is initiated whenever a guide rail encoding some tile type wins the bracket module. Once this happens, it grows into the external communication module, which grows datapaths in the direction of the neighboring macrotiles, and also tells the genome to do the same. Our first adjustment is encoding a “variable” instruction (from the original construction) into both datapaths such that they stop immediately before the boundary with the neighboring macrotile and wait for a signal to continue growing. Additionally, we augment the signal that tells the genome to start propagating so that it also initiates an added set of instructions in the initialization section of the genome that will begin the tiling process of the blocking protocol.
5.2.4 Receiver and piping

Once activated, the tiling instructions in the genome will start by growing to the upper eastern corner of the north face and placing a gadget which we refer to as the receiver. This gadget just waits for the boundary tiling to complete, similar again to the “variable” instruction. Once placed the datapath will then grow back down until it is a constant number of units above the center of the bottom face, i.e. just above the center of $\text{boundary}_D$. Here, it will grow the pipe intersection. Next, a pipe of constant length $\text{pipe}_D$ will grow down to the bottom face, along with an encoding of the scale factor of the macrotile $m$. Along $\text{boundary}_D$, this encoding will be used to seed counters in all four directions that will grow a distance of $\frac{m}{2}$ to the edges of the macrotile, i.e. the edges created by $\text{boundary}_D \times \text{boundary}_{\text{side}}$. In the space between the counters, filler tiles will attach by cooperation. One quadrant, however, will use special filler tiles that grow only a constant hardcoded number of steps. This will allow the outgoing genome and external communication to grow to the boundary first, and then cooperation will allow the filler tiles to continue, thereby rectifying a potential timing dependency. In the opposite quadrant, the same type of cooperation is used to allow the filler tiles to grow around an incoming genome and external communication, although no special tile is used here. This way, if the incoming datapaths aren’t present yet (as they possibly never will be) the filler tiles can fill out that space anyways, thereby blocking the datapaths if they ever do come in. Continuing on, the final row of each counter will initiate special signals in both directions perpendicular to the direction of growth, causing special tiles to be placed at the four bottom corners of the macrotile. The layout of the counters and special signals on the bottom face of the macrotile is illustrated in Figure 5.5.
Additionally, from the pipe intersection, a pipe\textsubscript{side} will grow out for each side direction. It will grow independently for a small constant number of steps, and then will be tethered to the counter on boundary\textsubscript{P} by a small, constant-width additional strip of tiles so that it grows right up to boundary\textsubscript{side} without overgrowing it. If the neighbor in the side direction has already differentiated, the pipe\textsubscript{side} might overlap with the signal that grows out of the pipe in the neighboring macrotile at exactly one tile location. However, we can design the signal such that it cooperates with the pipe\textsubscript{side} such that pipe\textsubscript{side} completes with the same structure and exposed glues regardless of whether the signal is present or not.

5.2.5 Growing the side faces

Each of the special tiles at the four bottom corners then grow up a constant height pole. Each pole starts growth of a constant width strip along the bottom of each boundary\textsubscript{side}, as seen in Figure 5.6. These strips must be careful to keep growing whether the incoming genome and external communication have grown through or not. Again, this is done by allowing the filler tiles to grow independently, or by letting them cooperate with special glues on the datapaths if the datapaths are present and initially block the filler tiles.

Once the filler tiles make it to the middle of the boundary, as designated by a special tile placed at the end of the counters, the tiles will wait (enforced by cooperation) for the corresponding pipe\textsubscript{side} to grow up to the boundary. Once present, the tiles can grow over the end of the piping and continue on the other side. However, they must wait again for the outgoing genome and external communication. This waiting is enforced by using a constant number of hardcoded tiles after passing the piping that must then cooperate with the tiles of the outgoing information in order to continue on. Once the two datapaths have also come
Figure 5.5: For this example, we make it so that the scale factor only needs to be encoded into the bottom face so that it knows how far to grow outwards to get to each side face. Each pipe that grows outwards in the N,E,S,W directions is tethered to the bottom face so that it also grows exactly up to each side face. Each black tile in the corner then grows up a pole (as seen on the left in Figure 5.6) that begins the growth of the bottom strip of each side face.
in, the tiles can continue all the way to the other edge. Since the cooperation will happen left-to-right and bottom-to-top, we can enforce that the upper right most tile will be the last placed.

Of the poles that initiated these bottom strips, the north western pole will also start growing a path around the top of all four strips (in the opposite direction that the strips themselves are growing). This path will also use generic tiles that turn around the corners and run along the top of these strips. Once the path has grown all the way around one loop, it will lift up one unit in the “up” direction and continue around again and again in a spiral assembly pattern. Since the first loop relies on cooperation with the bottom strips, we know that all four bottom strips must have completed by the time the path makes its first full loop. The spiraling path should make the pattern seen in Figure 5.7 on the north face. The final path around the macrotile in the uppermost ring should be calculated to begin right next to where the receiver was placed in the previous steps. This will cause the path to loop around and hit the receiver, signaling that the bottom face and all four side faces are completely tiled, and the macrotile is ready for differentiation.

5.2.6 Differentiation and activation

Once the receiver has been signaled, a path of tiles will grow back to the pipe intersection and into the loc_{center} location from the “up” direction. Placing this tile signifies that the macrotile has officially differentiated under our representation function \( R \). From here, for any direction \( \text{dir} \) with non-intercepted pipes, if the neighboring macrotile hasn’t differentiated, then tiles will attach within pipe_{dir} until they come out in the neighboring macrotile. Then, these tiles will activate the halted outgoing genome and external communi-
Figure 5.6: The bottom strip of each side boundary $\text{boundary}_{side}$. Growth starts from the special tiles on the left. Generic filler tiles grow either (a) around the incoming genome and external communication or (b) over the slots to block the incoming datapaths from coming in later. The special tile in the middle of the bottom connects to the end of the piping (thereby preventing the filler tiles from growing over it). A constant number of hardcoded tiles then count over to the slots designated for the outgoing genome and external communication. The filler tiles must then wait for these datapaths to come in before they can continue on. The tile in the upper-rightmost corner is guaranteed to be the last placed.

cation datapaths to continue growing into the neighboring macrotile. As for any $\text{pipe}_{dir}$ that was intercepted, this means that the neighboring macrotile has already differentiated, and tiles don’t need to / won’t attach within $\text{pipe}_{dir}$ past the $\text{intercept}_{dir}$ gadget. However, the hole that $\text{intercept}_{dir}$ leaves in $\text{pipe}_{dir}$ allows for diffusion in the “up” direction. If the macrotile in the “up” direction has not already differentiated, tiles can therefore diffuse in from above and will grow through $\text{pipe}_{dir}$, out of $\text{intercept}_{dir}$, and along the genome to signal that it (and the “up” external communication datapath) can continue growing in the “up” direction. If the neighbor in the “up” direction has already differentiated, then there is no need for these to continue growing in that direction, and the currently growing macrotile will become a constrained subspace anyways, not allowing new tiles to attach regardless.

The tiled boundary of neighboring macrotiles can prevent external communication datapaths from correctly propagating into non-differentiated macrotiles. However, because the boundary tiling only occurs after a guide rail has left the bracket, the external communication datapath would be unable to affect the tile type that the neighboring macrotile would eventually differentiate into. In other words, the macrotile has already essentially chosen a
Figure 5.7: Once the bottom strip has of each side face has completely filled out, one of the corner tiles starts the growth of the rest of the side faces by growing a one tile wide path around the top of each bottom strip. Each bottom strip can be certain to have completely by the time one loop has been made. Then, the path steps one in the “up” direction (through cooperation with the first tile in the path) and starts another path. This continue until the path reaches the receiver in the upper right corner of the side face on which the first path started. Once this receiver is hit, all the blocking protocol is certain to have completed, and the macrotile is clear to differentiate.
Once the bottom strips are tiled, all the timing dependencies are rectified and the rest of the process is sequential.

representative tile type, meaning new incoming datapaths don’t necessarily have to reach the macrotile’s genome for the simulation to progress correctly.

### 5.2.7 Seed and representation functions

In addition to the augmentations already discussed, the universal simulator also requires slightly tweaked functions $\mathcal{R}$ and $S$. In our implementation, $S$ must generate the seed macrotiles $\sigma_T$ such that the genome includes the new instructions previously mentioned. This covers the instructions to grow the interception gadgets when propagating into a new macrotile, the special glues on the perimeter when passing through the boundary of a new macrotile, the initialization instructions to place the receiver once the bracket has finished, etc. The representation function $R$ has to be augmented (for every output of the $\mathcal{R}$ function) to ensure that macrotiles with an open pipe intersection map to empty space (regardless of bracket state) and macrotiles with a blocked pipe intersection (and processed bracket) map to the correct tile type $t \in T$ in the simulated system $\mathcal{T}$.
5.2.8 Correctness

Overall, the main concern of our implementation is making sure the timing of the components is correct. This is important because, if certain components aren’t completed when the macrotile differentiates, this may leave unintended diffusion paths open. A depiction of all the timing dependencies (and their resolutions) that are critical to the correctness of our augmented construction is shown in Figure 5.8.

Similarly, the resolution of timing issues of blocking protocols between neighboring macrotiles is equally important. However, we know that the only potential inter-macrotile collision points are in the incoming genome and external communications datapaths and the signal that grows out of the pipes to re-initiate the growth of these datapaths. While this can affect the end growth of pipe<sub>side</sub> and the tiling of the bottom strips of boundary<sub>side</sub>, cooperation (as we previously mentioned) can ensure that these subassemblies form correctly irrespective of the state of the adjacent macrotile location. The “up” and “down” boundaries don’t have this issue since there are not adjacent boundaries being tiled. There are no other potential collision points between the blocking protocols of two neighboring macrotiles.

From here, we can prove that diffusion paths through a macrotile (from one side to another) exist only when the macrotile maps to empty space under $R$. Using this, we can then prove that paths through non-differentiated macrotiles can be strung together to make a diffusion path in $\mathcal{U}_T$ to macrotile locations that map to unconstrained space in $\mathcal{T}$.

**Lemma 1.** Given a macrotile $L$ and two directions $dir_A, dir_B \in \{N, E, S, W, U, D\}$ such that $dir_A \neq dir_B$ and the neighboring macrotiles in those directions have not already differentiated, a diffusion path from a tile location in the neighboring macrotile $A$ in direction $dir_A$,
through only macrotile $L$ (and no others), to a tile location in the neighboring macrotile $B$
in direction $dir_B$ will exist if and only if macrotile $L$ has not already differentiated.

Proof. For the proof, we will instead focus on two conditional statements that combined are
logically equivalent to the Lemma 1.

The first statement we will prove is, “If macrotile $L$ has not already differentiated,
then there is a diffusion path from $A$ to $B$ through $L$.” The most constrained the problem
can be while the premise is still true is if macrotile $L$ is one tile away from differentiating.
By proving the diffusion paths still exist in this situation, we prove they exist if macrotile
$L$ is earlier in the differentiation process. Now, we can break the problem down into two
cases. Let’s start with $dir_A, dir_B \in \{N, E, S, W, D\}$. In this simple case, the diffusion path
through $L$ from $A$ to $B$ is just in the tube in the $dir_A$ direction, through $loc_{center}$, and
out the tube in the $dir_B$ direction. The more complicated case is when, without loss of
generality, $dir_A = U$ and $dir_B \in \{N, E, S, W, D\}$. In this case, we know that some neighbor
in a direction $C \in \{N, E, S, W, D\} \setminus dir_B$ has already differentiated (in order for $L$ to have
initiated the differentiation process). Therefore, $intercept_C$ was present to “cut” a hole
in $pipe_C$. With this, our path is now moving from macrotile $A$, through the open top to
macrotile $L$, into $pipe_C$ through $intercept_C$, through the $loc_{center}$, and out $pipe_B$.

The other statement we will prove is, “If macrotile $L$ has already differentiated, then
there is no diffusion path from $A$ to $B$ through $L$.” Doing the opposite of the last claim, we
will now look at the least constrained the problem can be while the premise is still true, right
after the tile in $loc_{center}$ of macrotile $L$ has attached. Again, we can break the problem down
into two cases. First, when $dir_A, dir_B \in \{N, E, S, W, D\}$, the pipes in these two directions

88
are guaranteed to not have been intercepted, since neither $A$ nor $B$ has differentiated in the premise of the lemma. Since the pipe intersection is also blocked and there are no other breaks in the tiling of the side and bottom faces, there is no diffusion path. In the more complicated case, when $\text{dir}_A = \text{dir}_B \in \{N, E, S, W, D\}$, the path can start through the open top of macrotile $L$. While some pipes must have necessarily been intercepted, we know by the premise of the lemma that the pipe in direction $\text{dir}_B$ was not. Since that pipe is no longer connected to the intercepted pipes due to the tile attachment in $\text{loc}_{\text{center}}$, there is no diffusion path.

Together, these two claims prove Lemma 1.

Lemma 2. Given an empty tile location $l$ in the simulated system $\mathcal{T}$ and the corresponding macrotile $L$ in the simulator $\mathcal{U}$, tiles can attach within macrotile $L$ in the simulator if and only if tile location $l$ is not in a constrained subspace in the simulated system.

Proof. We again break the problem into two claims that together are logically equivalent to Lemma 2.

First, we will prove, “If tile location $l$ is not constrained, then tiles can attach within macrotile $L$.” By the premise, there must be a diffusion path in the simulated system $\mathcal{T}$ from infinitely far away to the tile location $l$. Therefore, there must be a series of macrotiles in the simulator $\mathcal{U}$ from infinitely far away to the macrotile $L$. Since these macrotiles all must map to empty space under the representation function $R$, none of them must have already differentiated. By Lemma 1, we know that each of these non-differentiated macrotiles has a path through it connecting all 15 pairs of directions. These mini-diffusion paths are guaranteed to lined up, since the pipes are designed to line up between adjacent macrotiles,
and can therefore be linked together to comprise a longer path. Therefore, there must be a diffusion path in the simulator $U$, comprised of these mini-diffusion paths through each macrotile concatenated together, from infinitely far away to any empty location within the macrotile $L$, thereby allowing tiles to attach in $L$.

Next, we will prove, “If tile location $l$ is constrained, then tiles cannot attach within macrotile $L$.” By the premise, the tile location $l$ is within a constrained subspace. By definition, there must be a constraining subassembly surrounding tile location $l$. In the simulator $U$, all of the tiles from this constraining subassembly must be represented by already differentiated macrotiles. Since a diffusion path in the simulator $U$ from infinity to the macrotile $L$ must pass through one of the macrotiles representing a tile in the constraining subassembly, and since we know that already differentiated macrotiles have no diffusion paths between neighbors in any two different directions by Lemma 1, then it must necessarily be the case that no diffusion path from infinity to the macrotile $L$ exists, and tiles can therefore not attach within the macrotile $L$.

Together, these two claims prove Lemma 2.

In showing that the blocking protocol can properly simulate the spatial constraint and doesn’t interfere with the dynamics of the modules from the original construction, we know that combining the original modules and the blocking protocol gives us an augmented construction that is capable of simulating any system within the Spatial aTAM. With the addition of the slightly modified seed generation function $S$ and representation function generator $R$, this augmented construction provides an intrinsically universal tile set for the Spatial aTAM, thereby proving Theorem 3.
6 Conclusion

To close out the thesis, we will go over the results we presented, their impacts, and future directions this research can take.

6.1 Summary of results and their impacts

Throughout this thesis, we looked at three different mathematical models of self-assembly: the Geometric Thermodynamic Binding Networks model, the Flexible Tile Assembly Model, and the Spatial abstract Tile Assembly Model.

To start off, we introduced the GTBN and showed that it was capable of universal computation via a method that is similar to a method commonly used in computational aTAM systems. This tells us two major things. First, it tells us that the lack of computational ability in the original TBN model, or at least the difficulty in its implementation if it is possible, is not a result of the lack of assembly process in the model. Rather, it provides strong evidence that the lack of a geometric constraint or any spatial aspect is the cause. Secondly, it tells us that computational designs within aTAM systems can be slightly augmented such that fully assembled computations can have favorable properties when viewed from a thermodynamic stability perspective.

Next, we introduced the FTAM and showed that it was capable of building a general set of polycube outlines in three dimensions. This is important because it gives evidence that, if we were able to implement FTAM dynamics in a real system, that system would be powerful enough to build this general set of shapes. In addition, we also provided some
discussion about the limitations on shape building, i.e. chirality, complex vertices, and multiple edge frames. The FTAM also provides a foundation for furthering the study of how local interactions between tiles can influence the global orientation of assemblies.

Finally, we introduced the Spatial aTAM and showed that it is intrinsically universal. This is important because it shows that an interesting computational result from the base 3D aTAM holds even a diffusion constraint is taken into consideration that makes the model more realistic albeit more complex.

6.2 Open problems and future work

We now discuss some open problems that our work presents, to give the reader an idea of how this work can be extended in the future.

Starting with the GTBN model, a future research direction could be following the connections between other geometric variations of the aTAM and the GTBN. Specifically the Polygonal TAM and Polyomino TAM mentioned in Section 2.4 would provide natural transitions into an exploration of results in the GTBN using non-square shaped geometric monomers. Additionally, work can be done in the traditional TBN to show a way to compute in the model or that it is impossible. If possible, this method would have to utilize dynamics different than those in typical computational aTAM systems that simply encode Turing machine transitions into tile types. This result would give more clarity as to the effects that certain differences in the model definitions have on computational ability.

In the FTAM, more work can be done to refine the properties of shapes that cause them to be either possible or impossible to build deterministically. Right now, we know symmetric polycube outlines with no reconfigurable vertices and exactly one edge frame
are possible to build deterministically. In addition, we know that asymmetric shapes are impossible to build deterministically, albeit, a change in the definition of what it means to “build” a shape could nullify this and remove the technicality that arises from the chirality of shapes in the FTAM. However, we have no more categorization of even other polycube outlines, much less the set of all possible shapes. This refinement would help further pin the capabilities and limitations of shape building in the FTAM.

In the Spatial aTAM, we showed that the diffusion constraint can be simulated by also utilizing the diffusion constraint in the simulator. However, it would also be interesting to know if it can be simulated without invoking the diffusion constrain in the simulator. In this case, the simulator would have to compute when a subspace has been cutoff, so that it knows not to differentiate macrotiles in the representative space in the simulator. Proving this result would be interesting because it would essentially tell us if the 3D aTAM can simulate the Spatial aTAM. We already know the reverse is true, given that the original construction from [17] never cuts off a subspace, and so it assembles the same whether in a diffusion constrained environment or not. Therefore, proving a simulator could compute the diffusion constraint would show that the models are computationally equivalent, while proving the inverse would show that the diffusion restricted model (Spatial aTAM) is actually more computationally powerful than the traditional model (3D aTAM) in some sense.

We hope that the results presented here will provide a solid foundation for research into these and many other questions, and continue to lead to greater understanding of the process of self-assembly. We hope that this will ultimately result in the design and physical implementation of even more powerful and diverse systems built via self-assembly.
Bibliography


