Universal Computation Using Self-Assembling, Crisscross DNA Slats

Jackson S. Bullard
University of Arkansas, Fayetteville

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

Part of the Other Computer Sciences Commons, and the Theory and Algorithms Commons

Citation

This Thesis is brought to you for free and open access by the Computer Science and Computer Engineering at ScholarWorks@UARK. It has been accepted for inclusion in Computer Science and Computer Engineering Undergraduate Honors Theses by an authorized administrator of ScholarWorks@UARK. For more information, please contact scholar@uark.edu.
Universal Computation Using Self-Assembling, Crisscross DNA Slats
Universal Computation Using Self-Assembling, Crisscross DNA Slats

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

By

Jackson Bullard
University of Arkansas
Bachelor of Science in Computer Science, 2023

May 2023
University of Arkansas

This thesis is approved for recommendation to the Honors College

______________________________
Dr. Matthew Patitz
Thesis Director

______________________________
Dr. Yong Wang
Committee member

______________________________
Dr. Lu Zhang
Committee member
Abstract

I first give a brief introduction to formal models of computation. I then present three different approaches for computation in the aTAM. I later detail generating systems of crisscross slats given an arbitrary algorithm encoded in the form of a Turing machine. Crisscross slats show potential due to their high levels of cooperativity, so it is hoped that implementations utilizing slats are more robust to various growth errors compared to the aTAM. Finally, my software converts arbitrary crisscross slat systems into various physical representations that assist in analyzing their potential to be realized in experiments.
THESIS DUPLICATION RELEASE

I hereby authorize the University of Arkansas Libraries to duplicate this thesis when needed for research and/or scholarship.

Agreed

Jackson Bullard

Refused

Jackson Bullard
ACKNOWLEDGEMENTS

I want to thank my advisor, Dr. Matthew Patitz, for always making me think a little harder.
TABLE OF CONTENTS

Abstract .................................................................................. ii
Acknowledgements ...................................................................... iv
Table of Contents ....................................................................... v
List of Figures ........................................................................... vii
List of Tables ............................................................................... xii
1 Introduction ............................................................................. 1

2 Formal Models of Computation ................................................. 2
  2.1 The Turing Machine .......................................................... 2
  2.2 The Abstract Tile Assembly Model (aTAM) ........................... 3
  2.3 The Kinetic Tile Assembly Model (kTAM) ............................. 3
  2.4 Polyominos in Self-Assembly .............................................. 4

3 Simulating Turing Machines in the aTAM ............................... 8
  3.1 Simulation with the aTAM ............................................... 8

4 Simulating Turing Machines using Polyominos ....................... 23
  4.1 Basic Turing Machine Simulation ...................................... 23
  4.2 Rightward Tape Growth .................................................. 31
  4.3 Leftward Tape Growth ..................................................... 31
  4.4 Slat Construction ............................................................ 39
    4.4.1 Input Slats .............................................................. 39
    4.4.2 Symbol-Copy Slats .................................................. 42
    4.4.3 Boundary Slats ....................................................... 43
    4.4.4 Transition Slats ....................................................... 46

5 Analysis .................................................................................. 51
  5.1 Slat Complexity .............................................................. 51
  5.2 Assembly Size ............................................................... 52
  5.3 Physical Considerations ................................................... 53
    5.3.1 scadnano .............................................................. 53
    5.3.2 oxView ................................................................. 53
6 Conclusion .................................................................................. 57
Bibliography .................................................................................. 58
LIST OF FIGURES

Figure 2.1: Depiction of how a tile in the aTAM can be implemented using four strands of DNA (courtesy of [1]). In each of the double helices, one of the strands exposes an unbound portion of itself; this produces the aTAM’s glues. ................................. 4

Figure 2.2: A single DNA crisscross slat. It is currently bound to nothing. ................................. 5

Figure 2.3: (a) Individual slats are separated by the black lines. The gray connector slats extend under the red body slats all the way to the other side of the body. I.e., each connector slat is connected to all four body slats. (b) Some macrotiles have both input signals come from their bottom half, but other patterns exist. ................................. 6

Figure 2.4: A step-by-step walk-through of the process of a macrotile assembling. (a) begins with two macrotiles to the west, and the southern one already has its output assembled. (b) and (c) show the northern macrotile assembling its output, which cooperates with the output from the southern macrotile. (d) through (f) show the body of the new macrotile assembling. Finally, (g) and (h) show the growth of the new macrotile’s output slats. ................................. 7

Figure 3.1: A complete assembly of the simulation of a Turing machine using the improved aTAM approach. The simulated Turing machine is used throughout this paper to showcase the various approaches and is described in Table 3.1. ................................. 8

Figure 3.2: Single-tile seed from which the assembly will grow. ................................. 10

Figure 3.3: Two tiles attached: a boundary growth tile to the left and an input tile to the right. ................................. 10

Figure 3.4: The entire input row. ................................. 11

Figure 3.5: An example showcasing how cooperativity enforces desired growth behavior. The bottom row of tiles must assemble before the top row can be completed because the bottom row has horizontal double-strength glues; the top row does not, so each tile in it relies on one glue from its western neighbor and one glue from the tile to its south. ................................. 12

Figure 3.6: Exactly one tile (left-most) in the input row has a northern strength-2 glue. This is the only place at which the next row can begin growing. We can see the only place capable of growth is the green circle; a tile there can bind to two glues by using one glue to the west and one to the south. ................................. 12

Figure 3.7: Growth of the first computational row. ................................. 13
Figure 3.8: Partial growth of an even computational row.

Figure 3.9: An example of a row where no operation was performed; the pink tile, which appears directly above the second head tile, represents a no-op. The appropriate transition is actually carried out in the last row shown.

Figure 3.10: The assembly’s progress as three transitions and two no-ops occur.

Figure 3.11: After reaching the end of the input, the state changes to zeros, indicating an even number of “1” symbols has been read. The simulated Turing machine now overwrites every input symbol with “0” symbols.

Figure 3.12: The whole assembly after the entire input has been read.

Figure 3.13: The first no-op tile for a left-moving transition (uppermost pink tile).

Figure 3.14: The process of overwriting the rightmost “1” symbol.

Figure 3.15: The simulated Turing machine has reached its halting state and left all zeros on the tape.

Figure 3.16: The entire assembly after the computation has completed. No more growth will occur.

Figure 3.17: Single seed for the improved design.

Figure 3.18: An odd-row head tile preparing for a rightward transition.

Figure 3.19: The resulting head tile attached. The star indicates the growth of a transition tile (white), while the lack of one indicates the growth of a head tile with the corresponding configuration.

Figure 3.20: Left transitions using the improved approach.

Figure 3.21: The final assembly using the improved approach.

Figure 3.22: The first computational row is all that is needed to simulate reading the entire input.

Figure 3.23: The second (and final) computational row writes the new values to each simulated tape cell.

Figure 4.1: A seed assembly composed of four vertical slats. Each vertical slat is 8 domains long, but only the upper four domains are available for binding.

Figure 4.2: The pink seed macrotile and its gray connector slats. The northeast “output” is not enough by itself for any vertical slats to attach with stability ($\tau = 4$). The northwest output may be considered a “double-strength” connection since it alone is enough for growth.

Figure 4.3: The same connector slats as Figure 4.2 with the pink macrotile slats removed for visualization purposes. Notice there is a total of four horizontal slats present.
Figure 4.4: The next macrotile begins to attach. Like every macrotile, it will consist of four vertical slats, each of which is exactly eight domains in length.

Figure 4.5: The entire input row. From top left to bottom right, we see a pink edge macrotile, a blue $BLANK+$ macrotile, a red even state, a “1” symbol macrotile, a white “0” symbol macrotile, a yellow “1” symbol macrotile, two consecutive “0” symbol macrotiles, another blue $BLANK+$ macrotile, and another pink edge macrotile.

Figure 4.6: Two identical rows stacked on top of each other.

Figure 4.7: The underlying connector slats for the input row shown in Figure 4.5 with the macrotiles hidden from view.

Figure 4.8: The connector slats from Figure 4.7 with the positions of macrotiles highlighted. The circled areas indicate valid positions for slat attachment.

Figure 4.9: The beginning of the growth of the first computational row. So far, only the boundary tile and a $BLANK$ tile have assembled.

Figure 4.10: The underlying connector slats for the portion of the assembly depicted in Figure 4.9. Slats highlighted with a blue outline indicate the output of the blue macrotile in the new row (the empty outline indicates positions of future slats). Those highlighted in red are the output of the head macrotile from the input row. The red circle indicates the area at which the outputs of the two tiles cooperate.

Figure 4.11: The first computational row assembled. So far, the system has simulated the Turing machine determining the parity of the number of occurrences of “1” symbols.

Figure 4.12: Growth of the second computational row has begun. The arrows indicate how the new boundary tile cooperates with the head tile from the row below.

Figure 4.13: The final assembly after finishing the simulation. The third row assembles, overwriting every input symbol with “0” symbols. A final row assembles on top to show the final contents of the tape.

Figure 4.14: A portion of an assembly for a Turing machine that immediately moved and needs to continue moving its head right (southeast in terms of the physical assembly). One “frontier-tooth” has attached (appears dark orange).

Figure 4.15: The underlying connector slats for the assembly pictured in Figure 4.14. The arrow shows which tile is outputting the slats.
Figure 4.16: A second tooth has attached, which exposes enough domains for horizontal output slats to stably bind to the assembly.

Figure 4.17: The underlying connector slats for the assembly pictured in Figure 4.16. The arrow indicates the slats on which it overlays are the output of the two frontier teeth.

Figure 4.18: Slats for the first tile in the new row begin to attach. Note: the nascent boundary tile isn’t directly above the previous one, but rather a cell over (to the right).

Figure 4.19: Because row growth now proceeds right-to-left (southeast to northwest), we encounter a no-op head macrotile (the northeast red macrotile).

Figure 4.20: The tape head continues moving right, so the next row requires another pair of frontier teeth.

Figure 4.21: The head tile signals the edge tile to widen the tape with a “grow” signal.

Figure 4.22: A growth $BLANK+$ tile assembles and presents a double-strength glue to its northwest, thereby widening the tape by one cell. To the southeast, a no-op head tile assembles, ready to be picked up and properly advanced to the left in the next row.

Figure 4.23: The new edge tile assembled. This process may continue as long as the tape head attempts to move left.

Figure 4.24: The southern corner of the partial assembly from Figure 4.13. Arrows indicate the outputs of macrotiles and are labeled with their respective output slat lengths in terms of number of domains.

Figure 4.25: The middle pink macrotile is the odd-row eastern edge macrotile, and the rightmost is the even-row variant. The extra long output of the odd variant of the eastern $BLANK+$ macrotile is also shown.

Figure 4.26: The even west edge macrotile and its northeastern outputs.

Figure 4.27: The four types of transition tiles necessary for each transition that moves the tape head to the right.

Figure 4.28: Additional right-moving transition tiles to respond to $BLANK+$ inputs.

Figure 4.29: The four types of transition tiles necessary for each transition that moves the tape head to the left.

Figure 4.30: Additional left-moving transition tiles to respond to $BLANK+$ inputs.

Figure 5.1: scadnano representation of the first three rows of the final assembly from Figure 4.13.
Figure 5.2: PolyominoTAS/scadnano assembly comparison using the single-macrotile seed assembly. .......................... 55
Figure 5.3: oxView visualization of the assembly shown in Figure 5.2 after relaxation. .................................................. 56
| Table 3.1: | Transition table of example Turing machine used throughout the rest of this paper. The start state of the machine is $even$, and the halting state is, appropriately, $halt$. | 9 |
| Table 4.1: | Transition table describing a Turing machine whose tape head moves right forever; used to display slat system capabilities for rightward tape growth. The start state is $right$, and it has no accessible halting state. | 32 |
| Table 4.2: | Transition table describing a Turing machine whose tape head moves left forever; used to display slat system capabilities for leftward tape growth. The start state is $left$, and it has no accessible halting state. | 32 |
1 Introduction

Algorithmic self-assembly is a relatively nascent field that has seen significant growth over the past few decades. Within it, there is a multitude of areas of study, most of which can be parsed into two categories: theoretical and experimental. My work establishes a theoretical approach to generate sets of polyominos capable of universal computation. Furthermore, I provide a complete software pipeline from abstract polyomino designs to simulated molecular dynamics using the scadnano API [2] and oxView [3, 4].

In Chapter 2, the concept of formal models of computation is introduced. It provides a brief description of Turing machines, the abstract Tile Assembly Model (aTAM), the kinetic Tile Assembly Model (kTAM), and polyominos in self-assembly. Chapter 3 details the precursor to this paper’s main result. Multiple approaches to universal computation in the aTAM are presented, each building upon the last. My main work is presented in Chapter 4. The text of these two chapters is accompanied by an extensive use of figures for two reasons: (1) to aid in understanding and (2) the work lends itself rather nicely to pictorial representations. Finally, the rest of the software workflow is introduced in Chapter 5. Physical considerations that are important for experimental planning will also be discussed there.
2 Formal Models of Computation

2.1 The Turing Machine

The Turing machine (named after computer scientist and mathematician Alan Turing) can be thought of as an abstraction computer scientists use to explore what it means to “compute”. Turing machines embody algorithms, and they give us a concrete way to analyze these algorithms while remaining, at their core, simple. A Turing machine consists of an internal state, an infinitely long tape, and a read/write head that allows it to read and modify one tape cell at a time. With these three simple components, Turing machines can compute any algorithm that today’s conventional computers can.

A particular Turing machine can be defined as a 6-tuple $M = \langle Q, q_0, q_H, \Sigma, \Gamma, \delta \rangle$.

1. $Q$: A finite, non-empty set of states. The Turing machine will always “occupy” exactly one of these states at a time.

2. $q_0$: The Turing machine’s start state; this is the state in which the machine is initialized.

3. $q_H$: The machine’s halt state; if the machine enters this state, it stops and performs no further computation.

4. $\Sigma$: The machine’s input alphabet. For the examples used in this paper, $\Sigma = \{0, 1\}$ will be used.

5. $\Gamma$: The machine’s tape alphabet. $\Gamma = \Sigma \cup \{b\}$, where $b$ is the blank symbol. For the systems presented in Chapter 3, $b$ equals the underscore character (“_”); the systems in Chapter 4, however, represent $b$ as “BLANK” for clarity in examples.

6. $\delta$: The transition function of the Turing machine. This defines the behavior of the machine or the algorithm that it represents. It is defined as the function $\delta : (Q \setminus \{q_H\}) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$. Essentially, $\delta$ is a partial function that receives as input the state that the machine is currently in and the symbol that it is reading from the tape. It outputs the next state to which the machine transitions, the new symbol the machine writes on the tape at the head position (i.e., it overwrites the symbol it was reading as input to $\delta$), and the direction in which to move the tape head (either left or right).
The *configuration* of the Turing machine is defined as the state it occupies, the contents of its tape, and the position of its tape head. A configuration can be thought of as a “snapshot” of all of the machine’s variables that define its current step in its computation.

2.2 The Abstract Tile Assembly Model (aTAM)

The abstract Tile Assembly Model (aTAM) [5] is a simple, formalized model that is useful for self-assembly design and analysis. A tile system in the aTAM is defined by an ordered triple $T = (T, \sigma, \tau)$. The elements of $T$ are the building blocks of the aTAM: square tiles with glues on their sides. An example of a tile can be seen in Figure 3.2. Unique tile types are defined by the glues assigned on their sides and the strength of each glue. Tiles will only attach in places where the glues on their sides match up with the glues on adjacent tiles. These tiles can be implemented physically as DNA structures, shown in Figure 2.1 [1]. The seed assembly, $\sigma$, is simply the beginning assembly from which growth must originate. It is composed of tiles in $T$ and can consist of one or more of these tiles.

In order for a new tile to attach to the growing assembly (i.e., the group of tiles connected thus far), the total strength of connecting glues must reach a threshold, typically referred to as the system’s “temperature”, or $\tau$. For example, in a system where the temperature equals 2 ($\tau = 2$), at least two sides with glues of strength 1 each or a single side with a glue of strength 2 must connect. Enforcing this type of cooperation ($\tau \geq 2$) between tiles grants the power of algorithmic assembly [6].

2.3 The Kinetic Tile Assembly Model (kTAM)

The aTAM is a useful theoretical framework with which to think about algorithmic self-assembly, but it alone completely ignores some glaring issues with reality. In physical experiments, DNA sequences don’t have to perfectly match to attach to an assembly (at least temporarily). The kinetic Tile Assembly Model (kTAM) addresses the added randomness with which tiles can attach to growing assemblies, and this perspective opens up many areas for approaches that mitigate the many different possible types of erroneous growth [7]. This framework has led to greater accuracy in the computational modeling of physical systems and the development of “proofreading” techniques, which make self-assembling systems more robust to algorithmic errors [8].
2.4 Polyominos in Self-Assembly

Without delving too deeply into what types of errors arise in the kTAM, we predict they can be mitigated by increasing the number of matching glues required for permanent attachment, which is accomplished by increasing the temperature, $\tau$, of the system [7]. That is, we hope to decrease erroneous growths by increasing the required level of cooperativity. While error prevention measures exist in the aTAM, this paper gives a construction using crisscross slats [9].

Physically speaking, crisscross slats can be implemented as individual strands of DNA. Shown in Figure 2.2, we can abstractly think of slats as a subset of polyominos. A polyomino is a 2-dimensional shape that is formed by connecting one or more unit squares by their faces [10]. Tiles in the aTAM are technically $1 \times 1$ polyominos, which simply consist of a single square. The construction provided in Chapter 4 mostly utilizes $1 \times 8$ or $1 \times 12$ slats, but an edge case requires the use of a single $2 \times 4$ polyomino. Furthermore, the system has $\tau = 4$.

Each constituent square in a polyomino can have a red mark in it, which indicates there is an associated domain. A domain is abstractly similar to “glues” as explained in the aTAM, although “glues” will receive a new definition in our polyomino systems. Unique domains are implemented as unique sequences of nucleotides. In contrast to the aTAM, we differentiate a domain and its corresponding complement to which it can attach. In PolyominoTAS [11], a domain’s complement is indicated by the identical label with an asterisk appended to the end. In

![Figure 2.1: Depiction of how a tile in the aTAM can be implemented using four strands of DNA (courtesy of [1]). In each of the double helices, one of the strands exposes an unbound portion of itself; this produces the aTAM’s glues.](image)
the aTAM, tiles attached at edges with identical glues. The idea of a complement domain was implicitly implemented by the fact that tiles could not rotate, so the directional glues could only bind to those of the exact opposite direction (e.g., an eastern glue may only attach to an identical western glue). Polyominos assemble by overlapping each other (as opposed to just being adjacent). This means we abstractly consider a third dimension, and, by convention, we separate assembled slats in space into $z = 0$ and $z = -1$. In reality, the DNA strands are intertwined in a crisscross manner [9].

A useful abstraction concerning slat assemblies is the concept of “macrotiles” (Figure 2.3). While certain geometric limitations preclude a perfect one-to-one mapping between aTAM tiles and slat macrotiles, we can consider them to function in a very similar manner. A macrotile consists of body slats and underlying connector slats. The body slats are displayed in $z = 0$ (on top of the connector slats), and we typically assign a common color between body slats for a particular macrotile in order to help us identify what the macrotile represents. We need four body slats to compose a macrotile, but my design only implements two unique slats for a given body. In other words, for each macrotile, my software builds an “A” body slat and a “B” body slat. These attach in an $ABAB$ pattern.

Connector slats remain in $z = -1$ and are displayed in gray. Unlike the body slats, each of these is unique. In the design presented in this paper, two adjacent connector slats are analogous to a single glue on the side of an aTAM tile. We can think of the gross signal propagated by a single pair of slats as a “glue”. When specifying a macrotile’s construction, the notion of macrotile inputs and outputs is essential. A macrotile will begin growth from its input glues. After the body slats assemble from a total of four input (underlying connector) slats, its output slats will assemble to the body slats. Part (b) of Figure 2.3 clarifies
Figure 2.3: (a) Individual slats are separated by the black lines. The gray connector slats extend under the red body slats all the way to the other side of the body. I.e., each connector slat is connected to all four body slats. (b) Some macrotiles have both input signals come from their bottom half, but other patterns exist.

this concept. All input slats must be present in order to reach a sufficient level of cooperation ($\tau = 4$). Output slats can then branch from the body slats and become another macrotile’s input.

In the figure, the label “Output #1”, for example, is the glue for the macrotile’s northwest output. This glue is simply an abstraction; it helps us think about the sum of the eight domains exposed in gray (four per slat). Recall that each domain is furthermore an abstraction for an actual sequence of nucleotides. Finally, similar to the aforementioned output, we refer to “Input #2” as the macrotile’s southeast input. Figure 2.4 shows the step-by-step process of a macrotile’s slats attaching to an existing assembly.
Figure 2.4: A step-by-step walk-through of the process of a macrotile assembling. (a) begins with two macrotiles to the west, and the southern one already has its output assembled. (b) and (c) show the northern macrotile assembling its output, which cooperates with the output from the southern macrotile. (d) through (f) show the body of the new macrotile assembling. Finally, (g) and (h) show the growth of the new macrotile’s output slats.
3 Simulating Turing Machines in the aTAM

In order to demonstrate the algorithms in this paper for generating sets of DNA tiles, let us consider an example Turing machine. At a high level, the Turing machine will receive a bit string as input, detect whether there is an odd or even number of “1” bits, and then modify the string accordingly. If the number of “1” bits is even, all “1” bits should be overwritten with “0” bits. Else, all “0” bits should be overwritten with “1” bits. In either case, the resulting string will have the same total number of bits as the input string, but it will be either all “1” bits or all “0” bits. To make things concrete, Table 3.1 gives the transition table for this Turing machine.

3.1 Simulation with the aTAM

Because the aTAM is Turing-universal [5], we can construct an algorithm to produce a tile set that simulates any given Turing machine. Consequently, given a universal Turing machine (a Turing machine capable of receiving any other Turing machine as input and simulating it [12]), we can construct a tile set that simulates that Turing machine, thereby obtaining a computationally universal tile set.

In the course of my work, I have implemented three approaches to generating aTAM tile sets that simulate Turing machines. The defining differences
Table 3.1: Transition table of example Turing machine used throughout the rest of this paper. The start state of the machine is even, and the halting state is, appropriately, halt.

between these approaches lie in the amount of space consumed by the assembly and the complexity of the tile sets, which I will measure by the number of unique tile types. I will first present the simplest (intuitively speaking) system.

**Simple aTAM Simulation**

We will only consider the temperature 2 aTAM. Tile sets for Turing machine simulation are composed of four groups of tile types: input tiles, boundary tiles, symbol copy tiles, and transition tiles. Input tiles are necessary for providing input to the computation. Each tile is associated with a symbol in the input string, and each expresses its symbol with an associated glue. Input tiles are where growth begins, and the entire row is typically replaced by a DNA origami that provides the same input in physical experiments. In my design, the assembly is seeded with a single tile (Figure 3.2). The assembly begins by growing horizontally using double-strength glues. The rest of the input row will fill in to the east, and the boundary tile responsible for initiating upward growth attaches to the west. The following figures are simulations courtesy of PyTAS [13]. In all of the following examples, we will simulate the Turing machine described in Table 3.1 on the input $x = 10100$.

The number of black bumps on the edges indicates the strength of the glue, and the green circles indicate valid positions for a title to attach. Because the system has a temperature of $\tau = 2$, the only valid positions for new tiles are
Figure 3.2: Single-tile seed from which the assembly will grow.

Figure 3.3: Two tiles attached: a boundary growth tile to the left and an input tile to the right.

adjacent to strength-2 glues (as opposed to the upper strength-1 glue). Figure 3.4 shows the entire input row grown. Note that there is nothing enforcing this row to fill out before tiles are added in the next row. In Figure 3.5, however, it is easy to see that the input row is necessary for growth to continue using cooperativity. Since $\tau = 2$, each tile must attach to the assembly using glues whose strength sum to 2. This is necessary because, as tiles grow across the next row, they must “read” the tape symbols from the previous row. These symbols are encoded in the northern glues of the previous row and enforce what tiles can attach above.

Subsequent rows are essentially snapshots of the simulated Turing machine’s configurations in order. The tape is represented by symbol tiles, which are typically white (except for blank tiles) in the figures of simulated assemblies. The state of the Turing machine and the position of the tape head are stored in head tiles, which are shown as red.

In most cases, a symbol copy tile will grow in the next row in order to further propagate the symbol stored in the corresponding tape cell of the simulated Turing machine. However, the point of having any input at all comes into play when the tape head is reading one of these symbols. The resulting glue that encodes a combination of the simulated machine’s state and the symbol in the cell from which it is reading enforces a transition tile to grow in the next row. This is how
the system implements the transition table of the simulated Turing machine.

Growth of subsequent rows proceeds in a “zigzag” fashion. The first computational row (i.e., the first row following the input row) will begin growing from the west, and the second computational row will begin growing from the east. This pattern repeats for the length of the computation. Figure 3.6 shows how the growth of the first computational row is initiated. Special boundary tiles at both ends of the assembly are responsible for initiating this growth with their unique strength-2 glues. Because a Turing machine theoretically has access to an infinite tape, care must be taken to simulate this using a finite-size assembly. Therefore, the boundary tiles are also responsible for widening the work tape. Each time a new row is grown, the tape will also be widened by two tiles: one on each side of the tape. This is more than sufficient because the tape head can only move one cell per row.

In Figure 3.7, we see two red tiles. These are the positions at which the simulated tape head rests on the work tape. Because we can think of each row of the assembly as a snapshot of the simulated Turing machine’s configuration, we see the tape head moved right in the first step of the computation. Looking at the northern glues of the two head tiles, we see the machine began in the even state but transitioned to the odd state. This happened because it read a “1” on its tape cell, so it changes state to keep track of the parity of the number of “1” bits.

The only way to carry information through the assembly is through glues on the edges of tiles. As you can see in Figure 3.7, information about the tape head and machine state propagated to a tile diagonal of the original head position. This is done using a transition tile. In the figure, the transition tile can be found by looking at the tile directly north of the bottom head tile. Using its eastern glue, it informs the growth of the tile to the east, enforcing it to receive a head tile in the prescribed state at that position.

If we number the computational rows (starting at 1 with the first row follow-
Figure 3.5: An example showcasing how cooperativity enforces desired growth behavior. The bottom row of tiles must assemble before the top row can be completed because the bottom row has horizontal double-strength glues; the top row does not, so each tile in it relies on one glue from its western neighbor and one glue from the tile to its south.

Figure 3.6: Exactly one tile (left-most) in the input row has a northern strength-2 glue. This is the only place at which the next row can begin growing. We can see the only place capable of growth is the green circle; a tile there can bind to two glues by using one glue to the west and one to the south.
**Figure 3.7:** Growth of the first computational row.

**Figure 3.8:** Partial growth of an even computational row.

**Figure 3.9:** An example of a row where no operation was performed; the pink tile, which appears directly above the second head tile, represents a *no-op*. The appropriate transition is actually carried out in the last row shown.
ing the input row), it is helpful to think of them as even versus odd. Alternatively, we number all rows (including the input row) starting at 0. This obviously gives us the same numbering for every row, but one may be more comfortable to think about than the other, especially considering the input row is likely to be replaced by a DNA origami in reality. Figure 3.8 shows an even computational row that is partially assembled. Even computational rows fill in from east to west, while odd computational rows fill in from west to east. This presents a challenge in some cases for transition tiles. The transition tile we pointed out in Figure 3.7 uses its eastern glue to inform the next tile’s growth; however, Figure 3.8 shows a row whose growth begins from the east and moves west. If the next transition of the Turing machine would require that the head moves to the right (and in this case, it does), the head tile needs to fill in before the transition tile is in position for that row. In this simple design, there is no way for the head tile to know what state it ought to be in or that there should even be a head tile there at all (an improvement to this behavior is discussed in the next section).

This physical limitation necessitates no-op tiles (short for “no operation”), an example of which can be seen in Figure 3.9. In the figure, the no-op tile appears pink and is directly above the second head tile. These tiles simply carry state and head position information up to the next row. Rows with no-op tiles can be considered exact copies of the row directly preceding them. In the following row, the appropriate transition tile and corresponding head tile correctly appear. The assembly essentially required an extra row to simulate that operation. The example Turing machine (Table 3.1) repeatedly moves right as it reads the entire input before performing any other operations, so a no-op will occur on every even row. The next transition can be seen in Figure 3.10.

Figure 3.10: The assembly’s progress as three transitions and two no-ops occur.
Eventually, our example Turing machine will reach the end of its input. When this happens, it should transition into its penultimate state. Which state it transitions to depends on the parity of the number of “1” symbols it saw in the input. If there was an odd number of “1” symbols, the machine transitions into the *ones* state, indicating it will overwrite each symbol of the input with a “1”. In our example, however, the input is $x = 10100$. Because there is an even number of “1” symbols, our example assembly simulates the machine transitioning into the *zeros* state. This transition can be seen in Figure 3.11, and the entire assembly up to this point can be seen in Figure 3.12.

![Figure 3.11](image)

**Figure 3.11**: After reaching the end of the input, the state changes to *zeros*, indicating an even number of “1” symbols has been read. The simulated Turing machine now overwrites every input symbol with “0” symbols.

Figure 3.13 shows the no-op tile for a left moving transition. We can also see in Figure 3.14 the rightmost “1” symbol being overwritten with a “0” symbol. The green lines at the bottom indicate the original values, and the upper green lines indicate the values at that particular step of the computation. Finally, Figure 3.15 shows the final result of all zeros on the tape, and Figure 3.16 shows the entire final assembly.

For the sake of brevity, I will keep the analysis regarding aTAM constructions simple. After all, these are not the main result of this paper. More in-depth analysis is given in Chapter 4 regarding the crisscross slat constructions. For this example, the number of unique tile types required is 59.
Improved aTAM Simulation

The first improvement we can make to the tile set is the removal of no-ops. This means always simulating exactly one step in the Turing machine’s computation on each row of the assembly. Figure 3.17 shows the single-tile seed for this design, which notably includes an asterisk next to the Turing machine’s state in the glue label. This asterisk indicates this start state, start symbol pair ought to be resolved by an attaching transition tile. The absence of the asterisk, on the other hand, enforces the attaching tile to be a head tile.

Because of the generic “left” and “right” directional glues on the eastern and western sides of tiles, every tile “knows” the parity of the row to which it has assembled. Transition tiles can leverage this knowledge to either pass state information to an intra-row neighbor or to the tile in the next row. For example (Figure 3.18 may be helpful), if the current head tile has just assembled on an odd row (which fills in west to east), there are two possibilities that depend on whether the transition moves the tape head left or right. It is easy for the head to move left; we handle the transition just the same as the previous design by allowing the even row (which fills in east to west) to assemble a transition tile atop the head tile. The next head tile then simply attaches to the west of the transition. However, if the transition requires that the tape head moves right, the head tile enforces its following neighbor to be a transition tile. A head tile will then attach in the next row, as shown in Figure 3.19.

Figure 3.20 shows the analogous leftward transitions. When trying to decipher in what direction a row fills, it is helpful to look at the symbol-copy (white) tiles. The angle bracket points in the direction of assembly for that row. While this design can result in a much smaller assembly (compare Figures 3.16 & 3.21),
Figure 3.13: The first no-op tile for a left-moving transition (uppermost pink tile).

it greatly increases the number of unique tiles in the system. For this example, the number of unique tile types required is 116. The following optimized approach, however, excels in both metrics.
Optimized aTAM Simulation

The optimized approach takes advantage of one simple concept: tiles can simply pass their new state to the neighboring tile if the row is growing in the same direction as the tape head is moving. Figure 3.22 illustrates this. When the tape head finally changes direction, the same transition tiles from the original design are perfectly sufficient. Figure 3.23 shows the next and final row of the assembly. In addition to greatly decreased assembly sizes, this approach generates fewer tiles than the original, only requiring 56 unique tile types for this particular example.
Figure 3.15: The simulated Turing machine has reached its halting state and left all zeros on the tape.
Figure 3.16: The entire assembly after the computation has completed. No more growth will occur.

Figure 3.17: Single seed for the improved design.

Figure 3.18: An odd-row head tile preparing for a rightward transition.
Figure 3.19: The resulting head tile attached. The star indicates the growth of a transition tile (white), while the lack of one indicates the growth of a head tile with the corresponding configuration.

Figure 3.20: Left transitions using the improved approach.
**Figure 3.21:** The final assembly using the improved approach.

**Figure 3.22:** The first computational row is all that is needed to simulate reading the entire input.

**Figure 3.23:** The second (and final) computational row writes the new values to each simulated tape cell.
4 Simulating Turing Machines using Polyominos

4.1 Basic Turing Machine Simulation

Transitioning from thinking in terms of the aTAM to polyominos can be difficult. To complicate things further, there are certain geometric hindrances to creating a one-to-one conversion from an aTAM assembly to the corresponding polyomino assembly. As mentioned in Section 2.4, crisscross slat systems can take advantage of high levels of cooperativity [9]. In our aTAM systems, cooperativity was limited to two ($\tau = 2$). The following polyomino examples will utilize $\tau = 4$.

Figure 4.1 shows the single macrotile assembly used as the system seed, and Figure 4.2 shows the connector slats after they have attached. Figure 4.3 shows these underlying connector slats while hiding the overlapping macrotile slats. Remember, the concept of two layers of slats utilizing a third dimension is not exactly accurate, but it is a helpful intuition and is how PolyominoTAS [11] simulates these systems (see Section 2.4). Underlying ($z = -1$) slats will appear gray and can be obscured by overlapping ($z = 0$) slats, which can be any color except gray. Because $\tau = 4$, four domains must bind for a slat to attach to the assembly. Recall that slats at $z = 0$ can only bind to slats at $z = -1$ and vice versa. Furthermore, slats in the $z = 0$ layer are vertically oriented, while slats in the $z = -1$ are oriented horizontally.

Thinking in terms of converting an aTAM system to a corresponding DNA slat system, we can consider the gray, underlying connector slats to be responsible for carrying the “glue” signals from their colorful macrotiles to adjacent portions of the assembly. In Figure 4.2, we see four slats exposed to the northwest and two to the northeast. We can consider outputs consisting of two slats the unit of “glue” strength, where we need two of these units for a stable binding, much like our aTAM examples. Another pair of slats are required to cooperate with the northeast connection for a new macrotile to bind. The northwest connection, however, possesses double the strength. At this point in the assembly, this is the only place at which growth can initiate. Figure 4.4 shows two slats attaching at this position. These partially form the next macrotile in the input row.

Figure 4.5 shows the complete input row. In physical experiments, this would likely be replaced by a DNA origami that exposes adapters in the place of the east-facing connector slats. These connector slats inform the growth of the next row, much like the aTAM systems mentioned in Section 3, by carrying information about the Turing machine’s configuration. Just as before, we are simulating the Turing machine described in Table 3.1 on the input $x = 10100$. Comparing Figure
Figure 4.1: A seed assembly composed of four vertical slats. Each vertical slat is 8 domains long, but only the upper four domains are available for binding.
Figure 4.2: The pink seed *macrotile* and its gray connector slats. The northeast “output” is not enough by itself for any vertical slats to attach with stability ($\tau = 4$). The northwest output may be considered a “double-strength” connection since it alone is enough for growth.

Figure 4.3: The same connector slats as Figure 4.2 with the pink macrotile slats removed for visualization purposes. Notice there is a total of four horizontal slats present.
Figure 4.4: The next macrotile begins to attach. Like every macrotile, it will consist of four vertical slats, each of which is exactly eight domains in length.
Figure 4.5: The entire input row. From top left to bottom right, we see a pink edge macrotile, a blue $BLANK^+$ macrotile, a red even state, a “1” symbol macrotile, a white “0” symbol macrotile, a yellow “1” symbol macrotile, two consecutive “0” symbol macrotiles, another blue $BLANK^+$ macrotile, and another pink edge macrotile.
4.5 to Figure 3.4 (the aTAM input row), the polyomino assembly is obviously a bit more involved. First, it is slanted and staggered. This can cause some difficulty in deciphering which tiles represent the same simulated cell in subsequent rows. Figure 4.6 shows two identical rows stacked on top of each other. If we replaced each macrotile with an aTAM tile, the assembly would be a perfect 2 × 5 rectangle. In Figure 4.5, all five colors of macrotiles are present. Like the aTAM, red macrotiles indicate the location of the simulated tape head. White and yellow macrotiles indicate tape cells containing the symbols “0” and “1”, respectively. Blue macrotiles indicate blanks, which are now encoded as “BLANK” for clarity (the previous aTAM examples used underscores). Finally, the pink macrotiles are special boundary tiles that are responsible for both “vertical” (bottom-left to top-right) and horizontal (tape-widening) growth. In this design, the tape only grows when necessary. This reduces the size of the final assembly.

Figure 4.7 shows the unobscured connector slats for the input row. Figure 4.8 is provided for clarity and also illustrates the areas capable of assembly growth, both of which are double-strength glues. The next row’s growth begins in Figure
Figure 4.7: The underlying connector slats for the input row shown in Figure 4.5 with the macrotiles hidden from view.

4.9, and we also see an example of how output slats from two separate macrotiles cooperate to facilitate growth. To be precise, the southeast output of the new BLANK macrotile cooperates with the northeast output of the head macrotile to facilitate the continued growth of the row. Although not pictured in the figure, the new BLANK macrotile will provide output to the next row through northeast output slats.

Figure 4.11 shows the entire first row of computations. We see the macrotile directly above the starting head position no longer hosts the head and simply contains a “1”. The head traveled across each tile that appears red and is now resting on the eastern-most BLANK macrotile. The next macrotile is the pink boundary tile, which is again responsible for initiating the growth of the next row. Figure 4.13 shows the final result of the simulation.
Figure 4.8: The connector slats from Figure 4.7 with the positions of macrotiles highlighted. The circled areas indicate valid positions for slat attachment.
4.2 Rightward Tape Growth

Rightward tape growth (i.e., southeast growth of additional macrotiles), presents a challenge for the physical nature of slat connections. In order to create a double-strength glue, a macrotile would need to provide output slats in the same position that it is receiving input. It is not possible for both required connector slats to bind to the same position in the body slats (it’s a double helix, not a triple, after all). It is also not possible to know ahead of time that the machine will need access to more rightward tape cells. Figure 4.14 shows a new polyomino: the frontier tooth (dark orange). This polyomino is still a single, 8-domain strand of DNA in reality, but it attaches to the assembly in the shape of a “U”. Once two adjacent frontier teeth attach (as seen in Figure 4.16, two more connector slats may join. This creates the necessary double-strength glue for growth to continue, which is shown in Figure 4.18.

4.3 Leftward Tape Growth

Unlike rightward tape growth, leftward tape growth does not run into similar geometric problems and can therefore be considered quite a bit simpler. Table
Figure 4.10: The underlying connector slats for the portion of the assembly depicted in Figure 4.9. Slats highlighted with a blue outline indicate the output of the blue macrotile in the new row (the empty outline indicates positions of future slats). Those highlighted in red are the output of the head macrotile from the input row. The red circle indicates the area at which the outputs of the two tiles cooperate.

<table>
<thead>
<tr>
<th>Start State</th>
<th>Start Symbol</th>
<th>End State</th>
<th>End Symbol</th>
<th>Head Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>right</td>
<td>0</td>
<td>right</td>
<td>0</td>
<td>R</td>
</tr>
<tr>
<td>right</td>
<td>BLANK</td>
<td>right</td>
<td>BLANK</td>
<td>R</td>
</tr>
</tbody>
</table>

Table 4.1: Transition table describing a Turing machine whose tape head moves right forever; used to display slat system capabilities for rightward tape growth. The start state is right, and it has no accessible halting state.

<table>
<thead>
<tr>
<th>Start State</th>
<th>Start Symbol</th>
<th>End State</th>
<th>End Symbol</th>
<th>Head Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>1</td>
<td>left</td>
<td>1</td>
<td>L</td>
</tr>
<tr>
<td>left</td>
<td>BLANK</td>
<td>left</td>
<td>BLANK</td>
<td>L</td>
</tr>
</tbody>
</table>

Table 4.2: Transition table describing a Turing machine whose tape head moves left forever; used to display slat system capabilities for leftward tape growth. The start state is left, and it has no accessible halting state.
Figure 4.11: The first computational row assembled. So far, the system has simulated the Turing machine determining the parity of the number of occurrences of “1” symbols.
Figure 4.12: Growth of the second computational row has begun. The arrows indicate how the new boundary tile cooperates with the head tile from the row below.
Figure 4.13: The final assembly after finishing the simulation. The third row assembles, overwriting every input symbol with “0” symbols. A final row assembles on top to show the final contents of the tape.
Figure 4.14: A portion of an assembly for a Turing machine that immediately moved and needs to continue moving its head right (southeast in terms of the physical assembly). One “frontier-tooth” has attached (appears dark orange).

Figure 4.15: The underlying connector slats for the assembly pictured in Figure 4.14. The arrow shows which tile is outputting the slats.
**Figure 4.16**: A second tooth has attached, which exposes enough domains for horizontal output slats to stably bind to the assembly.

**Figure 4.17**: The underlying connector slats for the assembly pictured in Figure 4.16. The arrow indicates the slats on which it overlays are the output of the two frontier teeth.
Figure 4.18: Slats for the first tile in the new row begin to attach. Note: the nascent boundary tile isn’t directly above the previous one, but rather a cell over (to the right).

Figure 4.19: Because row growth now proceeds right-to-left (southeast to northwest), we encounter a no-op head macrotile (the northeast red macrotile.)
Figure 4.20: The tape head continues moving right, so the next row requires another pair of frontier teeth.

4.2 gives the transition function for a Turing machine whose only goal is to infinitely move its tape head left. Figure 4.21 shows a head tile that is reading a \textit{BLANK+} tile (not pictured) from the row below. It signals the adjacent edge tile to widen the tape. Figure 4.22 shows a \textit{BLANK+} tile assembling above the edge tile. \textit{BLANK+} macrotiles behave identically to \textit{BLANK} symbol macrotiles for the purpose of computations, but it has a special function regarding the physical assembly of the system. In this case, it is responsible for growing an edge tile to its northwest, which widens the tape by one. Figure 4.23 shows the result. In general, \textit{BLANK+} macrotiles provide a signal to the assembly that enables certain slats responsible for growth to attach before reaching the very edge of the assembly. This is useful for situations where the tape needs to be expanded, and some situations where growth of the next row is initiated needs this advanced warning.

4.4 Slat Construction

I wrote my software [14] for generating DNA slats for Turing machine simulation using Python 3.10. The process for generating slats is broadly split between four types of slats: input, boundary, symbol copy, and transition. The following description of the algorithm will assume $\tau = 4$ and an \textit{ABAB} pattern for macrotile body slats for all constructions.

4.4.1 Input Slats

The portion of the algorithm that generates input slats creates a number of macrotiles equal to the number of characters in the input (representing those
Figure 4.21: The head tile signals the edge tile to widen the tape with a “grow” signal.

Figure 4.22: A growth $BLANK^+$ tile assembles and presents a double-strength glue to its northwest, thereby widening the tape by one cell. To the southeast, a no-op head tile assembles, ready to be picked up and properly advanced to the left in the next row.
Figure 4.23: The new edge tile assembled. This process may continue as long as the tape head attempts to move left.

Figure 4.24: The southern corner of the partial assembly from Figure 4.13. Arrows indicate the outputs of macrotiles and are labeled with their respective output slat lengths in terms of number of domains.
characters) plus three edge macrotiles (an east and a west \textit{BLANK}+ and a west \textit{EDGE}). Notice, there is one more tile in the input row than there are described here. The northwest boundary tile (pink) is a generic boundary tile that is also capable of assembly elsewhere during the computation. In our example of \(x = 10100\), it generates 8 macrotiles. For each macrotile, six slats total are generated: two body (\(z = 0\)) slats (using the \textit{ABAB} pattern) and four output (\(z = -1\)) slats. The domains of the body slats are unique to the macrotile and its corresponding output slats. The northwest glue of each input-row macrotile is double strength (and therefore utilizes all four output slats), while the northeast glue for each is single strength.

Although the body slats for any macrotile in the system are always 8 domains in length (for \(\tau = 4\)), the length of output slats can either be 8 or 12 domains long. Figure 4.24 shows, in general, how output slats cooperate between rows. For input-row macrotiles (and, more generally, even row macrotiles), their output slats assemble exclusively to the northern half of the macrotile. Generally, the southern pair of these output slats is exposed as the glue to the northwest, and the northern pair is exposed as the glue to the northeast. In the input row’s case, however, the northern pair is extra long (12 domains) and additionally provides input to the northwest to create a double-strength glue by binding their middle four domains to their macrotile’s body slats. Figures 4.2 and 4.3 show exactly what this means. This double-strength output is essentially hard-coded for growth of each input-row macrotile. The actual input to the computation is expressed by the northwest outputs of each of these macrotiles. This output, along with input from one of its sides, tells the cell in the following row what it should be (usually either a symbol-copy or transition macrotile).

4.4.2 Symbol-Copy Slats

The symbol-copy slats can be considered the simplest out of the four types. For every symbol in the tape alphabet, we only need to make two symbol-copy macrotiles: one for even rows and one for odd rows. The difference between the two lies in where their outputs are. For odd rows, their output slats assert themselves to the northeast and southeast. The red macrotiles in Figure 4.24, while not symbol-copy macrotiles, exemplify this output pattern. For even rows, they output to the northwest and northeast. Looking at the same figure, the top-right pink and blue tiles display this output pattern, but are also not actually symbol copy tiles.

For either type, their northeast output carries information regarding what symbol is in that simulated tape cell, and their other output is fed to their neighboring macrotile that is next to be assembled. This output is a simple signal that essentially indicates a lack of a transition occurring on the assembling tape cell, and it is necessary to both enforce which parity (even/odd) of macrotile to assem-
ble and provide the necessary cooperativity for continued growth. We’ll hereafter refer to these signals as “left” and “right” for even and odd rows respectively. The inputs to symbol-copy macrotiles are simply the symbol they represent (from the previous row, fed to their southwest input) and the aforementioned generic row-assembly signal (i.e., “left” or “right” depending on macrotile parity). For odd macrotiles, this input comes from the northwest, and, for even macrotiles, it comes from the southwest.

4.4.3 Boundary Slats

The construction of boundary slats is the most tedious of the four slat types. My design leaves no way around enumerating the process to construct each one, although is possibly an area for improvement. In order to maintain a bearable level of brevity, even versus odd input/output slat locations follow the conventions mentioned earlier unless explicitly said otherwise.

We first make an even and an odd version of the pink-colored west edge macrotile. The even version is responsible for initiating next-row growth. For input, it sees its own odd version in the row below and the generic “left” signal. Its output must be double-strength, so it uses four slats to output to the northeast. Since it is the edge of the assembly by definition, it does not output to the west at all. Instead, the lower pair of output slats reaches across a length of two macrotiles (Figure 4.26). The first half of these extra long (12 domain) output slats and the upper output slats serve as input to the odd version of this macrotile. The latter half of the extra long slats is simply the generic “right” signal. Use of these extra long slats is necessary because the double-strength input to the next row occupies space in the macrotile that is otherwise used for output slats to attach. The odd variation of this macrotile simply outputs its presence to the next row and receives no input from the northwest.

The east edge is very similar in function to the west edge. The even-row variation simply outputs a generic “left” signal and a signal to the next row to indicate its presence. It receives input from the odd variation of itself and from the odd variant’s neighbor’s output, which utilizes extra long slats in order to create a double-strength glue. The odd variation has no southeastern output since it is the edge of the assembly. It simply receives input from its even-row counterpart and the generic “right” signal. It outputs a signal to enforce the growth of its even variant in the next row, which must cooperate with the aforementioned extra long output slats from its neighbor. Figure 4.25 shows an example.

In order to avoid colliding input and output slats when utilizing double-strength glues to initiate next-row and tape-width growth, we buffer the edge macrotiles of the assembly with BLANK+ macrotiles. These act as blank tape cells for computational purposes (and therefore share the blue coloring), but they
Figure 4.25: The middle pink macrotile is the odd-row eastern edge macrotile, and the rightmost is the even-row variant. The extra long output of the odd variant of the eastern $BLANK^+$ macrotile is also shown.
assist the growth of the physical assembly. The western $BLANK^+$ macrotile is simpler than its eastern cousin. The even variant simply receives and outputs the generic “left” signal in its same-row input/outputs. From the previous row, it sees there was an odd version of itself, and it provides a complementary output to the next row to enforce the growth of its odd version. The even version is nearly identical, although it uses the generic “right” signal and flips its inter-row input output glues (i.e., it reads an even version of itself from below and enforces the growth of the even version above).

The eastern version of the $BLANK^+$ macrotile is slightly more complicated. The even variant is simple enough; it inputs/outputs the generic “left” signal like before, and its inter-row input/output functions just like the western $BLANK^+$ macrotiles. The odd eastern $BLANK^+$ macrotile also receives and outputs the generic “right” signal as expected. However, because of the geometry of the eastern edge of the assembly, the odd version must utilize extra long output slats in order to facilitate next-row growth. Its intra-row output slats are extended by four domains (12 domains total) to reach the edge tile in the next row. Figure 4.25 shows this.

All of the previously mentioned macrotiles keep the assembly growing (and, by extension, the simulation computing) so long as the computation does not require any tape cells outside of those used for input. In order to handle Turing machines that may need more room, we must build tape-extension slats, which are briefly showcased in Sections 4.2 and 4.3. The simpler case is leftward tape growth. This requires two additional macrotiles. We create an additional even, west edge macrotile that functions in almost an identical manner to the vanilla even, west edge macrotile. The only difference is that, instead of signalling an odd, west edge macrotile to grow in the next row, it signals the next new macrotile: a west growth blank (Figure 4.22). This special $BLANK^+$ macrotile outputs a double-strength glue to the northwest to assemble a generic edge macrotile one space to the west (one cell left on the simulated tape). Its output to the next row simply acts as a generic $BLANK^+$ macrotile.

The other case is rightward tape growth. This requires one new macrotile and a new polyomino type. Examples can be seen in Figures 4.14 through 4.20. The new macrotile is an even edge “growth” macrotile. When a transition macrotile assembles on top of a $BLANK^+$ macrotile, it signals this new edge growth macrotile to assemble. This macrotile enforces its predecessor in the next row to become a $BLANK^+$ macrotile using its northeast output; its southeast output, however, is special. It uses extra long slats, which are necessary to create a double-strength glue that is two macrotiles away. Unfortunately, there is no way of knowing ahead of time if the tape needs to expand, so the input from the previous row to this growth edge macrotile occupies the space necessary to create a double-strength glue using its own outputs.
To work around this, we utilize “frontier teeth”. These can still be implemented by a single strand of DNA, but they bind in the shape of a “U”. Again, one of these can be seen in Figure 4.14. They are eight domains long, but their shape allows them to form something akin to half of a macrotile while binding to only two slats. This is possible because they are still binding at four domains. They then allow the assembly of two more output slats (of normal length) that form a double-strength glue alongside the growth edge macrotile’s output slats. This then assembles a generic edge macrotile, and the rest of the row can fill in normally, only it is now one macrotile (tape cell) wider.

4.4.4 Transition Slats

Transition slats quickly dominate the number of slats in the system as the complexity of the Turing machine grows. For every transition, we must make four macrotiles. If the transition’s start state is the machine’s blank symbol (BLANK in our examples), an additional two macrotiles must be constructed to handle transitions that occur on BLANK+ symbols. As long as the Turing machine’s transition table includes every possible start state/tape symbol pair, the number of transition macrotiles generated is equal to

$$4 \cdot |Q| \cdot |\Gamma| + 2 \cdot |Q|$$

where $Q$ is the set of states and $\Gamma$ is the set of tape alphabet symbols.

Transitions must be treated differently depending on whether they require the tape head to move right or left. For every transition that moves the tape head right, create the following transition macrotiles (see also Figure 4.27):
1. Fast odd-row transition: receives its start state from its western neighbor and its start symbol from the row below. It outputs its end state to its eastern neighbor and its end symbol to the next row. This type of transition macrotile always occurs between two transition tiles in the same row.

2. Signal-pickup, odd-row transition: receives the generic “right” signal from its western neighbor and its start state, start symbol pair from the row below. It outputs its end symbol to the next row and its end state to its eastern neighbor.

3. Generic even-row transition: receives its start symbol from the row below and its start state from its eastern neighbor. It outputs its start state, start symbol pair to the next row and the generic “left” symbol to its western neighbor.

4. Even-row no-op: receives its start state, start symbol pair from the previous row and the generic “left” signal from its eastern neighbor. It outputs the same start state, start symbol pair to the next row (thereby performing no operation) and the generic “left” signal to its western neighbor.

Furthermore, if the start symbol of the transition is the blank symbol, additionally create the two following transition macrotiles (see also Figure 4.28):

1. Odd-row eastern BLANK+ macrotile: receives its start state from its western neighbor and its start symbol (the blank symbol) from the row below. It outputs its start state, start symbol pair to the next row. It also signals its eastern neighbor, an edge macrotile, to widen the simulated tape. This occurs when the tape head has moved to the eastern boundary of the assembly and still must travel eastward.

2. Even-row western BLANK+ macrotile: receives its start state from its eastern neighbor and its start symbol (again, the blank symbol) from the row below. It outputs its start state, start symbol pair to the next row and the generic “left” signal to its western neighbor. This is essentially the tape head turning around after moving left, and it happens to have landed on a BLANK+ macrotile.

For every transition that moves the tape head left, create the following transition macrotiles (see also 4.29):

1. Fast even-row transition: receives its start symbol from the previous row and its start state from its eastern neighbor. It outputs its end state to its western neighbor and its end symbol to the next row.
Figure 4.27: The four types of transition tiles necessary for each transition that moves the tape head to the right.

Figure 4.28: Additional right-moving transition tiles to respond to \textit{BLANK}+ inputs.
Figure 4.29: The four types of transition tiles necessary for each transition that moves the tape head to the left.

2. Signal-pickup, even-row transition: receives its start state, start symbol pair from the previous row and the generic “left” signal from its eastern neighbor. It outputs its end state to its western neighbor and its end symbol to the next row.

3. Generic odd-row transition: receives its start state from its western neighbor and its start symbol from the row below. It outputs its start state, start symbol pair to the next row and the generic “right” symbol to its eastern neighbor.

4. Odd-row no-op: receives its start state, start symbol pair from the row below and the generic “right” signal from its western neighbor. It outputs its start state, start symbol pair to the next row and the generic “right” symbol to its eastern neighbor.

Furthermore, if the start symbol of the transition is the blank symbol, additionally create the two following transition macrotiles (see also Figure 4.30):
1. Even-row western $BLANK+$ macrotile: receives its start state from its eastern neighbor and its start symbol (the blank symbol) from the row below. It outputs its start state, start symbol pair to the next row. It also signals the neighboring edge macrotile to widen the simulated tape. This occurs when the tape head has moved to the western boundary of the assembly and must continue traveling westward.

2. Odd-row eastern $BLANK+$ macrotile: receives its start state from its western neighbor and its start symbol (the blank symbol) from the row below. It outputs its start state, start symbol pair to the next row. This macrotile differs from the rest of the transition macrotiles, however, because it utilizes an extra long pair of output slats in order to help facilitate next-row growth. This is necessary because it is taking place of the usual $BLANK+$ macrotile that would otherwise handle this. The first half of the extra long output provides the generic “right” signal to the neighboring edge tile. The second half simply provides the signal to assemble the next row’s eastern edge tile.

**Figure 4.30:** Additional left-moving transition tiles to respond to $BLANK+$ inputs.
5 Analysis

In this section, I discuss some key metrics by which we can judge my design. Unique tile complexity is important to measuring the efficiency of a tile system’s design. Additionally, a smaller assembly size is desirable: the less assembly required, the less time needed for assembly to take place and the simpler it is to measure the results of an experiment. Finally, I discuss some methods to analyze the physical reality of a slat design and provide a complete workflow from abstract slats to simulated molecular dynamics.

5.1 Slat Complexity

A key metric of the design of a tile system is the number of unique tiles or, in our case, polyominos. For each macrotile added to an assembly, a maximum of six unique slats are added: two body slats (assuming an $ABAB$ pattern) and four output slats. For the purposes of this analysis, I will not include the count of input macrotiles since they are typically replaced by a DNA origami.

1. **Boundary** macrotiles: My design generates a constant 11 boundary macrotiles and the single frontier tooth polyomino.
   (a) Total: $(11 \cdot 6) + 1 = 67$

2. **Symbol-copy** macrotiles: For every symbol in the tape alphabet (including the blank symbol), two macrotiles are generated.
   (a) Total: $2 \cdot |\Gamma| \cdot -6$, where $|\Gamma|$ is the size of the tape alphabet.

3. **Transition** macrotiles: For every transition in the Turing machine’s definition, four macrotiles are created. Additionally, two macrotiles are added for every state in the Turing machine to account for the additional $BLANK+$ behavior.
   (a) Total: $6 \cdot (4 + 3 + 2 + |Q|)$, where $|\delta|$ is the machine’s number of transitions, and $Q$ is the number of states.

We can therefore characterize the total number of slats generated for an assembly as

$$24 \cdot |\delta| \cdot +12 \cdot |Q| \cdot +12 \cdot |\Gamma| \cdot +67$$

The example Turing machine (described in Table 3.1) consists of the states $even$, $odd$, $zeros$, $ones$, and $halt$; however, the transition table (and, by extension, the
encoding given to the software) does not define any transitions that utilize $halt$ as a start state. We therefore do not count the $halt$ state when generating slats and obtain a count of $|Q| = 4$. According to Table 3.1, there are 12 transitions ($|δ| = 12$: one transition for each state, symbol pair). Finally, there are three unique symbols in the tape alphabet ($|Γ| = 3$: “0”, “1”, and “BLANK”). The number of slats generated in order to simulate this Turing machine is therefore represented by

$$24(12) + 12(4) + 12(3) + 67 = 439 \text{ slats}$$

The software technically generates an additional 48 slats (for our example) that assemble the input row, but, since this row is typically replaced by a singular origami, it is not included in the count. For the sake of being thorough, if $|x|$ is the length of the input string, the number of input row slats generated is

$$6|x| + 6(3)$$

5.2 Assembly Size

The simulation utilizes a similar approach to the “optimized” version of the aTAM simulations (i.e., it performs as many computations as possible as long as the direction of row growth matches the direction of the tape head). This allows potentially many individual operations to occur per assembly row. Furthermore, the tape grows horizontally only if the simulated machine attempts to move past what is immediately available. These two properties keep the number of rows small and the width of the assembly low.

Ignoring tape expansion, in the worst case (say, a Turing machine that oscillates the tape head left and right for every transition), the system simulates one operation per row; fortunately, this is a rather unrealistic example. Most computations require initially reading a chunk of the input (if not all of it), which means multiple consecutive transitions that move the head to the right. Then, as part of a computation, whether that means referencing a previous part of the input or overwriting some data, multiple consecutive left transitions must occur.

The true worst case scenario, however, is a Turing machine that exclusively moves in one direction. It is only during tape expansion that a row will simulate no operation whatsoever. In this case, the system simulates one operation every two rows. Again, this is rather unrealistic. While it is perfectly reasonable for a computation to require more space than that of its input, a Turing machine that exclusively moves in one direction is not utilizing the contents of its tape, which is where the Turing machine, as a computational model, derives its power. Typically, after a machine writes some data in the expanded portion of the tape,
it will move back to read some previous data, thereby utilizing the advantages afforded by simulating multiple equal-direction transitions in the same row.

5.3 Physical Considerations

5.3.1 scadnano

Now that we have established a theoretical approach to simulating Turing machines using DNA crisscross slats, we ought to analyze its physical properties. In order to accomplish this, my work utilizes scadnano [2], a browser-based tool that is useful for designing synthetic DNA structures. It is a valuable tool when considering more physically realizable characteristics of DNA structures. Using its Python API, I have developed software that converts arbitrary assemblies from PolyominoTAS [11] simulations to scadnano assemblies. The scadnano API also provides functionality to convert its assemblies into oxView [3, 4] assemblies. Figure 5.1 shows the first three rows of the final slat assembly from our example simulation.

For a clearer picture, Figure 5.2 shows the seed assembly and its corresponding scadnano representation. The pink body slats in (a) show up as the zigzagging vertical strands in (b), and the gray output slats are displayed as the straight horizontal strands. In this example, domains are five nucleotides long; however, DNA helices complete a full rotation every 10.5 base pairs. In order to maintain assembly stability, we must account for this non-integer periodicity by occasionally adding insertions to the strands. These insertions are essentially a single extra nucleotide that doesn’t significantly affect the binding behavior of the domain. In the presented design, horizontal strands follow a 5-5-5-6 pattern: three domains of length five and a single domain of length six.

5.3.2 oxView

In order to analyze the physical behavior of the simulated assemblies, my software utilizes the scadnano API to generate oxView assemblies [3, 4]. The browser-based tool simulates DNA as a small number of constituent molecular components. The simulations are more course-grained than those at the atomic level but are specifically tuned for DNA molecular dynamics. Since crisscross slats are a rather unnatural use of DNA’s binding properties, oxView’s live “relaxation” function is essential for analyzing the potential feasibility of slat systems. In short, relaxation simulates how the DNA moves according to the forces imposed by bound domains. It gives us an opportunity to see how likely slats are to stay assembled if they are already in place. Currently, my software is limited to assigning a single DNA sequence to every $z = 0$ slat and the complementary sequence to every
Figure 5.1: scadnano representation of the first three rows of the final assembly from Figure 4.13.
Figure 5.2: PolyominoTAS/scadnano assembly comparison using the single-macrotile seed assembly.

\[ z = -1 \] slat, but the code has a module that should make it easy to plug in a mapping between every domain label and its corresponding sequence assignment.

My software aims to bridge the gap between abstract design to fully-specified DNA experiments. It is my hope that the slat generation capabilities provide a robust theoretical foundation upon which physical experiments may be designed. Furthermore, my software, using the scadnano API, converts the simulated abstract assemblies to representations that provide insight to their physical reality. This implements a complete workflow from the description of a Turing machine to simulated molecular dynamics.
Figure 5.3: oxView visualization of the assembly shown in Figure 5.2 after relaxation.
6 Conclusion

Using the Turing machine as the foundation upon which base my design, I’ve shown how to generate computationally universal sets of crisscross slats that can be physically realized by strands of DNA. I also provide a complete workflow that translates polyomino assemblies to scadnano and oxView representations using the scadnano API. In short, I present tools that, with the help of PolyominoTAS, scadnano, and oxView, provide a complete workflow that transforms a description of a Turing machine into DNA strand designs and molecular dynamics simulations. It is my hope that this workflow can be used to realize wet-lab experiments that perform arbitrary computation using self-assembling, synthetic, single-stranded DNA.
Bibliography


