COMPUTATIONAL ASPECTS OF DNA SELF-ASSEMBLY SYSTEMS AT TEMPERATURE 1

by

Bahar Behsaz
B.Sc., Sharif University of Technology, 2010

a Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science

in the
Department of of Mathematics
Faculty of Science

© Bahar Behsaz 2013
SIMON FRASER UNIVERSITY
Summer 2013

All rights reserved.
However, in accordance with the *Copyright Act of Canada*, this work may be reproduced without authorization under the conditions for "Fair Dealing." Therefore, limited reproduction of this work for the purposes of private study, research, criticism, review and news reporting is likely to be in accordance with the law, particularly if cited appropriately.

APPROVAL

Name:	Bahar Behsaz
Degree:	Master of Science
Title of Thesis:	Computational Aspects of DNA Self-Assembly Systems at Temperature 1
Examining Committee:	Dr. Petr Lisonek Chair
	Dr. Ladislav Stacho, Professor, Department of Mathematics Simon Fraser University Senior Supervisor
	Dr. Ján Manŭch, Adjunct Professor, Department of Mathematics Simon Fraser University Co-Supervisor
	Dr. Luis Goddyn, Professor, Department of Mathematics Simon Fraser University SFU Examiner
Date Approved:	May 1, 2013

Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the "Institutional Repository" link of the SFU Library website (www.lib.sfu.ca) at http://summit/sfu.ca and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author's written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, British Columbia, Canada

revised Fall 2011

Abstract

In this thesis, we investigate the computational power of some variants of Winfree's abstract Tile Assembly Model (aTAM) at Temperature 1 [43]. Although aTAM at temperatures higher than 1 are proved to be Turing Universal, i.e. they can simulate an arbitrary Turing Machine [43], the computational power of aTAM at temperature 1 is still an open question. It is known that some modifications of aTAM are indeed Turing Universal at temperature 1 [11, 30]. In this thesis, we first show that two variants of aTAM, namely the Staged Tile Assembly Model and Step-wise Tile Assembly Model at Temperature 1, are also Turing Universal.

Next, we discuss the computational power of the self-assembly with triangular tiles and hexagonal tiles, respectively. We prove that these models can simulate arbitrary systems under aTAM and vice versa, and consequently, they have the same computational power as aTAM.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Self-assembly is a bottom-up process by which fundamental components autonomously coalesce into more complex patterns or structures without human intervention. More formally, *self-assembly* refers to the spontaneous formation of structures through a process that involves assembly of existing components, and it can be controlled by proper design of the components, the environment, and the interaction force [31]. Nature provides many examples: atoms react to form molecules, molecules react to form crystals and supermolecules. This phenomenon is a key concept for fabrication of desired nano-structures. Instead of building nano-scale objects molecule by molecule, molecular self-assembly enables the scientists to produce these objects by designing the molecules and letting them to spontaneously self-assemble useful structures.

DNA self-assembly, pioneered by Seeman [36], is one of the fruitful areas in the study of self-assembly systems, in which double and triple crossover DNA molecules have been designed to act as four-sided self-assembling nanoscale construction blocks or *tiles* that can bind together via a fundamental concept called *base pairing* [37]. In 1982 Seeman [36] proved experimentally the feasibility of practical implementation of self-assembling molecular tiles [36]. This method has made substantial progress over the past three decades. Regular arrays [44], polyhedra [17], fractal structures [19], curved three-dimensional vases [20], DNA tweezers [46], logic circuits [32], neural networks [33] and molecular robots [28] are a few examples of the many nano-structures successfully self-assembled in laboratory experiments. Some of the industrial motivations for DNA self-assembly include designing efficient computer chips, single molecule detection and achieving better in-cell diagnosis and treatment.

In accordance with their practical importance, self-assembly models have received increased theoretical attention over the last years. In 1998, Winfree [43] introduced the *(abstract) Tile Assembly Model* (aTAM or TAM) – which utilizes the idea of Wang tiling [42] – as a simplified mathematical model of the DNA tile self-assembly. Very briefly, a *tile* in TAM is a unit square with a "glue" on each side of it. The four pairing ends of DNA tiles are presented as four sides of the unit square. Each of the sides has a specific glue associated with it, representing different nucleotide sequences. This idea is illustrated in Figure 1.1. The bonding strength between different glue types can be different, and is often modeled as 0, 1 or 2 etc. A tile may attach to an existing structure if the sum of the bonding strengths of the bonds formed between the tile and the existing structure exceeds some threshold temperature τ. The self-assembly process starts from an initial *seed* assembly ψ, and proceeds by iteratively attaching copies of tile types to the growing structure one by one. This model is described more formally in Section 1.1.

Several fundamental problems in the field of tile assembly model have been studied. A few of these problems are the following. (1) *Shape fabrication measures*: given a shape S (where S is a finite, connected subset of $\mathbb{Z}\times\mathbb{Z}$) what is the minimum number of distinct non-*empty* tile types that uniquely self-assemble into S [29, 4, 2, 35, 23]? Since the total number of tile types dominates the cost (in money and time) of practical implementation, this measure is crucial in programming large complex molecular self-assembly systems. Another resource constraint that has been considered is the time required to build a shape S [1, 9, 34]. (2) *Shape complexity*: given a small number of tile types, what shapes can be constructed under TAM or related models [40, 24]? (3) *Error detection*: TAM makes two assumptions that in practical experiments may not hold: it is assumed that tiles never detach from an assembly and a tile *only* attaches to the growing structure when the binding strength exceeds the temperature. To address these practical issues in theoretical models, several techniques and tools are suggested [43, 39, 39, 14, 10]. (4) *Computational complexity*: given algorithmic essence of TAM, it is important to find the computational power of this model. It is shown that despite its deliberate over-simplification, the TAM is computationally very powerful. In fact, it is proved that TAM at temperature 2 or higher, is computationally universal, that is, it is able to simulate an arbitrary Turing machine [43]. Although significant amount of work has been done on temperature 1 self-assembly models [7, 11, 26, 30, 3, 15], the answer to the following question is still unknown: what is the computational power of deterministic planar positive-strength temperature 1 self-assembly? In this thesis, we are making further step in answering the above question. In particular, we will prove computational universality in two known self-assembly models, namely, Step-wise and Staged self-assembly models at temperature 1.

Besides the square tiles, experiments have been done to produce tiles of triangular and hexagonal shapes. For example, Liu et al [27] reported the construction of a DNA triangle tile composed of three four-arm junctions, also Ding et al [13] obtained a triangular DNA tile formed from some different forms of DNA molecules, and He et al [21] built a hexagonal DNA tile. This motivated the study of comparing limitations self-assemblies using the square, triangular and hexagonal tiles [25]. In particular, they designed shapes that can be assembled in one model, but not the others at temperatures 2 and 3. In the second part, of this thesis, we study the computational power of Self-Assembly systems with triangular and hexagonal tile shapes at temperature 1.

Thesis Layout. In the remainder of this chapter, we give a formal description of the abstract Tile Assembly Model and in Section 1.2 we describe some variants of aTAM that have been proposed through the past two decades. In Chapter 2 we present some known results on tile complexity and computational power of tile assembly models. In Section 2.1 we survey some tile complexity results in aTAM and two other tile assembly models, namely Step-wise tile assembly model and Staged tile assembly model. In Section 2.2 we discuss some known work on computational power of tile self-assembly models. In particular we present the results on Turing Universality of few self-assembly models.

In Chapter 3 we present our result on computational power of two tile assembly models, namely Step-wise tile assembly model and Staged tile assembly model. We show that at temperature 1 these two models are very powerful from computational point of view and are in fact capable of simulating an arbitrary Turing Machine on an arbitrary input. An extended abstract of the results in chapter 3 appeared in [7].

In Chapter 4 we study the impacts of using different tile shapes on computational of power of tile assembly systems at temperature 1. We show that at temperature 1 Triangular and Hexagonal tile assembly models are equivalent to aTAM from computational point of view.

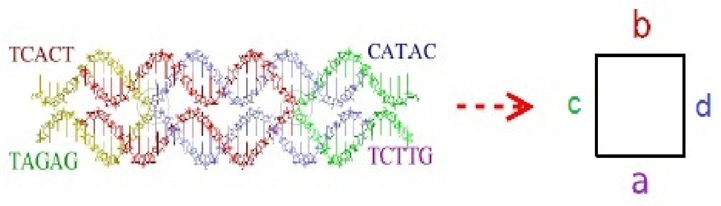


Figure 1.1: A DNA tile mapping to a tile type in TAM.

1.1 Definition of the Abstract Tile Assembly Model

In this section, we present a brief description of the tile assembly model based on Rothemund and Winfree [43]; for a more detailed description we refer the reader to [43]. We will be working on a $\mathbb{Z}\times\mathbb{Z}$ grid of unit square *positions*. The set of directions $D = \{N, E, W, S\}$ is composed of four $\mathbb{Z}\times\mathbb{Z}\mapsto\mathbb{Z}\times\mathbb{Z}$ functions such that:

$$N(i, j) = (i, j + 1),\ E(i, j) = (i + 1, j),\ S(i, j) = (i, j - 1) \text{ and } W(i, j) = (i - 1, j).$$

The inverse directions are defined naturally, for example $N^{-1}(i, j) = S(i, j)$, etc.

Let Σ be a set of *binding domains (glues)*. The set Σ contains a special binding domain *null* that represents no glue, in this thesis we also denote the glue *null* by $\_$. A *tile type*, t, is a 4-tuple $[\, t_N\, ,\, t_E\, ,\, t_S\, ,\, t_W\,] \in \Sigma^4$ indicating the associated binding domains on the north, east, south and west sides of t, respectively. We denote the glue on the side d ($d \in D$) of a tile t with t_d. Note that tile types are oriented, thus a rotated version of a tile type is considered to be a different tile type. Let T be the set of tile types. There is a special tile type *empty* $= (null, null, null, null)$ in T which represents an empty space, i.e., no tile has been place in that position. A *configuration* is a function C, mapping $\mathbb{Z} \times \mathbb{Z}$ to the set of tile types in T. Hence, $C(i, j)$ denotes the tile placed at the position (i, j) in the configuration C. We call the tiles which share a side with t, *neighbors* of t. Let a *structure* Struc(C) of a configuration be the set of positions that are not mapped to *empty* in C.

Under Tile Assembly Model a *tile assembly system* (TAS) is a 5-tuple $\mathbf{T} = \langle \Sigma, T, \psi, s_\Sigma, \tau\rangle$, where T is the finite set of tile types with binding domains from Σ and contains the tile *empty*, ψ is a set of configurations on T called set of *seed configurations*, s_Σ is a function which determines the bonding strength between glues in Σ, and τ is a threshold parameter called temperature. In this thesis we will omit parameters Σ and s_Σ when they are clear from the context.

A *strength function* $s_\Sigma : \Sigma \times \Sigma \mapsto \mathbb{N}$, satisfying $s_\Sigma(\sigma, \sigma') = s_\Sigma(\sigma', \sigma)$ and $s_\Sigma(\sigma, null) = 0$, measures the strength of interaction between binding domains. Two tiles abut on the sides with glues σ and σ' respectively, *bind with strength* $s_\Sigma(\sigma, \sigma')$. For example, many authors consider the *simple strength function*, $s_\Sigma(\sigma, \sigma') = 1$ if $\sigma = \sigma' \neq null$, and $s_\Sigma(\sigma, \sigma') = 0$ otherwise. In this thesis, we assume $s_\Sigma(\sigma, \sigma') > 0$ if $\sigma = \sigma'$ and $s_\Sigma(\sigma, \sigma') = 0$ otherwise. A tile t can be added to a configuration at position (x, y), only if the sum of the strengths of interactions between glues on sides of t that abut the already placed tiles at neighboring positions is at least τ (the temperature). Consequently, given a position (x, y), a direction $d \in D$ and a configurations C, we say there is a *bond* between positions (x, y) and $d(x, y)$ in C if and only if $s_\Sigma(C(x, y)_d, C(d(x, y))_{d^{-1}}) > 0$. That is to say, the strength function of the pair of glues on the abutting sides of the tiles placed at the two positions (x, y) and $d(x, y)$ is non-zero.

Given two configurations C and D, we define their union by $A = C \cup D$ to be the following map from $\mathbb{Z} \times \mathbb{Z}$ to $T \cup \{\infty\}$:

$$A(x,y) = \begin{cases} C(x,y) : \text{if } D(x,y)\text{=empty or } C(x,y) = D(x,y) \\ D(x,y) : \text{if } C(x,y)\text{=empty or } C(x,y) = D(x,y) \\ \infty : \text{otherwise} \end{cases}$$

We say that $C \cup D$ is a configuration if it is a map to T. Thus, self-assembly can be defined as relation between configurations on T. Let C and D be two configurations under a tile assembly system, say $\mathbf{T}$, such that $C = D$ except at a position (x, y), where $C(x, y) = empty$ and $D(x, y) = t$, for some $t \in T \backslash \{empty\}$. Then we write $C \rightarrow_{\mathbf{T}} D$, if and only if the sum of interaction strengths on the sides of t with tiles at neighboring positions reaches τ. In this sense, we can define the self-assembly process as a *sequence* of tiles that attach to the growing structure one by one under the self-assembly relation starting from the seed tile. Formally, the self-assembly process is the transitive closure of $C \rightarrow_{\mathbf{T}} D$, which we denote by $C \rightarrow^{+}_{\mathbf{T}} D$.

The seed configuration is the configuration where the We focus on a subset of configurations that are produced by the self-assembly process starting from seed configuration. In particular, a tile system $\mathbf{T}$ and the self-assembly relation $\rightarrow^{+}_{\mathbf{T}}$ define the partially ordered set of configurations, called *assemblies* of $\mathbf{T}$, denoted by $Asmb(\mathbf{T}) = \{A : \psi \rightarrow^{*}_{\mathbf{T}} A\}$. A tile system *uniquely assembles* A if for all $B \in Asmb(\mathbf{T})$, $B \rightarrow^{+}_{\mathbf{T}} A$. We say that a system is *deterministic* if for every $A, B \in Asmb(\mathbf{T})$, either $A \rightarrow^{*}_{\mathbf{T}} B$ or $B \rightarrow^{*}_{\mathbf{T}} A$, i.e., there is a unique sequence of assemblies producing the terminal assembly. Note that a deterministic system always uniquely assembles the terminal assembly. An assembly A is said to have *no binding domain mismatches* if for any two neighboring positions (x, y) and $d(x, y)$, where $d \in D$, such that $A(x, y) \neq empty$ and $A(d(x, y)) \neq empty$, we have $A(x, y)_d = A(d(x, y))_{d^{-1}}$. If the assembly process reaches a point when no more attachments are possible, the produced assembly is called *terminal* and is considered the output assembly of the system. We denote terminal assemblies of a system T by $\mathrm{Term}(T)$. For a finite deterministic system, we have $|\mathrm{Term}(T)| = 1$ and we define $T(i, j) = \mathrm{Term}(T)(i, j)$.

A *shape* S is a connected subgraph of the lattice induced by $V(S) \subseteq Z \times Z$. We say an assembly A *has shape* S if and only if $V(A) = V(S)$. For example, a shape S is an $N \times N$ square if there exists a position $(x_0, y_0) \in V(S)$ if and only if $x_0 \leq x < x_0 + N$ and $y_0 \leq y < y_0 + N$. We say an assembly A is *full* if for any two neighboring position (x, y) and $d(x, y)$ in $V(A)$, where $d \in D$, there is a bond between them in A. We say a tile system *uniquely produces* a full shape S, if the terminal assembly of the tile system is unique, full and has shape S. The smallest number of non-empty tile types and non-null glues required to assemble a given shape are called the *tile complexity* and *glue complexity* of the shape, respectively.

Given a deterministic assembly system $\mathbf{T}$, we say that g is an *input glue of a tile t at a position (x, y) via side d* if $t_d = g$ and when placing t into the growing structure it forms a bond with the tile at position $d(x, y)$ already present in the structure, and we say that g is an *output glue of a tile t at a position (x, y) via side d* of an assembly if $t_d = g \neq null$ and after placing t into the growing structure, side d of t is *exposed*, i..e., there is no tile at position $d(x, y)$ in the structure just after t was placed. Note that by this definition, input and output glues cannot be *null*, and since the system is deterministic, the input and output glues are well defined. Also, we call a set of tiles that are attached together a *supertile*.

Given a standard tile system $\mathbf{T} = \langle \Sigma, T, \psi, s_\Sigma, 1 \rangle$, and a configuration C of $\mathbf{T}$, the *backbone graph* of C, $G(C) = (V, E)$ is the subgraph of the square lattice whose vertex set is $V(C)$, and there is an edge between two vertices (x, y) and (x', y') if and only if there is a bond between $C(x, y)$ and $C(x', y')$ in C. Note that if the configuration C is also an assembly of $\mathbf{T}$, then its backbone graph is connected.

1.2 Other Models of Self-Assembly

Over the years new models and variants of TAM have been introduced and studied for both theoretical and experimental purposes. Further, we will see that adding some simple constraints and concepts to TAM, substantially improves efficiency and "power" of tile self-assembly models.

- **Staged Tile Assembly Model.**[12] Erik D. Demaine, et. al, presented the *Staged tile assembly model*, a generalization of the standard tile assembly model that captures the temporal aspect of laboratory experiment, and enables more flexibility in the design and fabrication of complex shapes using a small tile and glue complexity. In this model the tiles can be added gradually in a sequence of stages, and non-attached tiles can be washed away and removed at the end of each stage (in practice, this can be done by using a weight-based filter, for example). Also, there can be any number of bins, each containing tiles and/or assemblies that self-assemble as in the standard tile assembly model. During a stage, any collection of the following two operations are allowed: (1) add (arbitrarily many copies of) a new tile to an existing bin; and (2) pour one bin into another bin, mixing the contents of the former bin into the latter bin. In both cases, tiles that do not assemble into larger structures are removed at the end of the stage. These operations let us build intermediate terminal assemblies in isolation (i.e. in a separate bin) and then combine different terminal assemblies to form more complex structures. Two new complexity measures in addition to tile and glue complexity arise: the number of stages, or *stage complexity*, measures the time required by the operator of the experiment, and the number of bins, or *bin complexity*, measures the space required for the experiment. Note that, when both of these complexities are 1, the model is a equivalent to the regular tile assembly model.

- **Step-wise Tile Assembly Model.** [34] This model is the special case of Staged tile assembly model, in which only one bin is being used. Under Step-wise tile assembly model, each step can have a different tile set. Let $\{T_i\}_{i=1}^{k}$ be a sequence of finite sets of tiles, where T_i represents the tile set used for ith step of the experiment. In the first step, the seed tile is immersed into T_1, the terminal assembly is retained and the rest are filtered out. In the next step the assembled structure is placed in T_2, where it acts as a seed configuration. The assembly continues in this manner. Step-wise tile assembly model enables production of more efficient assemblies in terms of tile types at the price of the work of an operator [29]. The time required for the experiment is measured by the number of steps or *step complexity*.

- **Zig-Zag Tile Assembly Model.** [11] Informally, a Zig-Zag assembly is an assembly in which the assembly can grow only left to right (or right to left) up to the point at which a first tile is placed into the next row north, and after entering the next row, the direction of growth is reversed compared to the row below, see Figure 1.2.
  Temperature 2 Zig-Zag assembly system: A tile system $\Gamma = \langle T, 2, s \rangle$ is called temperature 2 Zig-Zag system if:

  1. The assembly sequence, which specifies the order in which tile types are attached to the system is unique, i.e. the system is deterministic.
  2. If the $i-1$th tile added to the assembly sequence is placed in an even row (counting from 0 starting with the row containing the seed tile) in position (x, y) of the grid, then the ith tile is either placed in position $E(x, y)$ or $N(x, y)$. And if the $i-1$th tile added is placed in an odd row, then the next tile to be added (ith tile in the sequence) is placed either in $W(x, y)$ or $N(x, y)$.

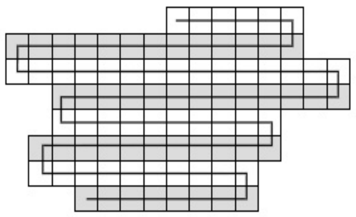


Figure 1.2: The direction of growth in a Zig-Zag system alternates between left and right.

- **3D Tile Assembly Model.** [11, 8] In this model the abstract Tile Assembly Model is extended to 3 dimensions by using unit cubes as the building blocks instead of 2 dimensional unit squares used in TAM, and adding glues to the top and bottom side of the cubes. In this model the domain of the mapping function explained in 1.1 is extended to $Z \times Z \times Z$ instead of a 2-dimensional grid in TAM. A tile attaches to any of the six sides of already placed tiles if the strength function at the corresponding position exceeds the temperature threshold τ.

- **Restricted Glue Tile Assembly Model (rgTAM).** [30] In comparison to TAM, here the range of the strength function s_Σ is $\{1, -1\}$ and there exists *exactly* one glue, say a, that satisfies the following statement,

  $$\exists a \in \Sigma : \forall b \in \Sigma, s_\Sigma(a, b) = -1.$$

  Similar to what described before, in any rgTAS a tile t can attach to an assembly at a specific position, if and only if the sum of strength function between glues on the abutting sides of the already placed tiles and t is at least equal to the temperature threshold τ.

- **Multiple Temperature Model.** [5] In the multiple temperature model, the integer temperature threshold τ is replaced with a sequence of integer temperatures $\{\tau_i\}_{i=1}^{k}$, called the *temperature sequence* of the tile system. In a tile system with k temperatures, assembly takes place in k phases. First, tiles are added to the seed tile as in the standard tile assembly model under temperature τ_1. When no more tiles can be attached, phase 2 starts and the temperature of the tile system switches to τ_2. Now, tiles can be added as in the standard tile assembly model under τ_2, and some tiles may also break off, if the strength of their bonds are strictly less than τ_2 and the assembly containing the seed tile is retained at the end of each phase. Once no more tiles can be added or removed, phase 2 is complete and phase 3 starts. The temperature is set to τ_3 and tiles are now added and removed under this new temperature. This process continues until phase k is complete.

- **Flexible Glue Model.** [5] In contrast to TAM, in this model unequal glues may bind together with non-zero strength. One can think of this by designing several sticky ends on the sides of the DNA tile molecule, while the binding between tiles are dependent to the number of matching glues on the abutting sides of the tiles.

- **Unique Shape Model.** [5] A tile system, T, under this tile assembly model, must uniquely assemble a shape S, that is to say the terminal assemblies produced by the tile system are all have shape S. In other words, contrary to TAM, T may have several different terminal assemblies (i.e. nondeterministic growth) but all these terminal assemblies are in shape S.

- **Kinetic Tile Assembly Model** [43] Comparing to the other models previously described, *Kinetic tile assembly model* is a more realistic model of tile assembly that is suggested by Winfree, to address some of the practical issues that are not considered in TAM. The assumption here is that a tile type can attach to any of the exposed glues on an assembly, at a rate proportional to its concentration, regardless of its strength of attachment. For more details see [45].

Chapter 2

Related Work

In theory, Winfree's abstract Tile Assembly Model is very powerful in the sense that relatively small sets of tiles may be used to simulate the universal Turing machines [11] or a cellular automata [43]. This suggests that a self-assembly system may be viewed as a program to build complex (computable) shapes [2]. Such an algorithmic view to self-asselmbly is interesting because it suggests that computer algorithms and programs should be able to capture all of the complex behaviours of self-assembly systems and help as a mean of designing efficient self-assembly systems. Also, theoretical principles of computer science can be translated to statements about objects in real world. Working under assumption that any tile system can be implemented physically, researchers have begun to explore the computational powers of these systems. In this section, we review some results on the self-assembly systems from the point of view of computational complexity.

2.1 Tile Complexity in Tile Based Self-Assembly Models

The algorithmic aspect of tile self-assembly systems declares that their ability to perform computation is directly related with their ability to build large and complex configurations by using small variety of tile types compared to the *size* of the constructed structure. Also, as mentioned earlier, reducing the number of tile types decreases the cost of practical experiments. Hence, one of the most popular questions that arises in the area is: given a specific shape, find the tile system with minimum number of tile types to self-assemble this shape. The decidability version of this question can be described as the following: given a shape A, a positive integer c and a threshold temperature τ, is there a tile system with

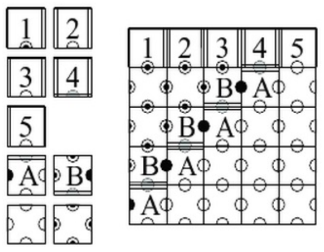


Figure 2.1: [35] A tile assembly system that uniquely assembles a 5×5 full square

less than c tile types which self-assembles A at the given temperature. Adelman et. al [2], proved the latter question is NP-complete. Then they asked, "What is the [tile] complexity of generating an $N \times N$ square with self-assembly?". The attempts to answer this question and its varieties led to a fruitful body of work in self-assembly studies and have had great impacts in understanding the computational aspects of self-assembly systems[2, 12, 41, 35, 5, 29]. In this section, we study some of these results.

2.1.1 Tile Complexity of Full Squares in the Abstract Tile Assembly Model

We start by stating some of the early results regarding temperature 2. By the end of this section, we will also show that the tile complexity of an $N \times N$ square at temperature 1 is quadratic in regard to N, while this measure can be much smaller for temperature 2.

Theorem 1. *[35] The tile complexity of an $N \times N$ full square at temperature 2 is at most $O(N)$.*

Proof Sketch. Figure 2.1 shows a tile system that uniquely assembles a 5×5 full square using only $N + 4 = 9$ tile types. In this figure, the seed tile is positioned at the top leftmost position of the square and the top border is formed with strength 2 bonds with tile types labeled with numbers. When the top border is finished, the tile types labeled A in the picture attach to the top border with the strength 2 glue on their north side and enter the next row. Then the tile types labeled B fill in the diagonal positions and unlabeled tiles bind to the existing assembly with the strength 1 glues. The described process for the 5×5 full square can be generalized to any positive integer N. □

Even a better bound is possible by combining the method mentioned in [35], to self-assemble a fixed width binary counter, with the construction mentioned in the proof of Theorem 1. A sketch of the construction for binary counter is mentioned in the next lemma.

Lemma 2. *[35] There exists a self-assembly system $\boldsymbol{T} = \langle T, s, 2\rangle$ that simulates a simple binary counter machine.*

Proof Sketch. The general idea of the proof is illustrated in Figure 2.2. The assembly contains seven distinct tile types. The borders are assembled with strength 2 bonds and the rest of the assembly is finished with strength 1 glues. Simulation of the action of incrementing is regulated by the location of the glue c in each row by controlling which "digits" are changing from 0 to 1 or vice versa as the assembly grows row by row. □

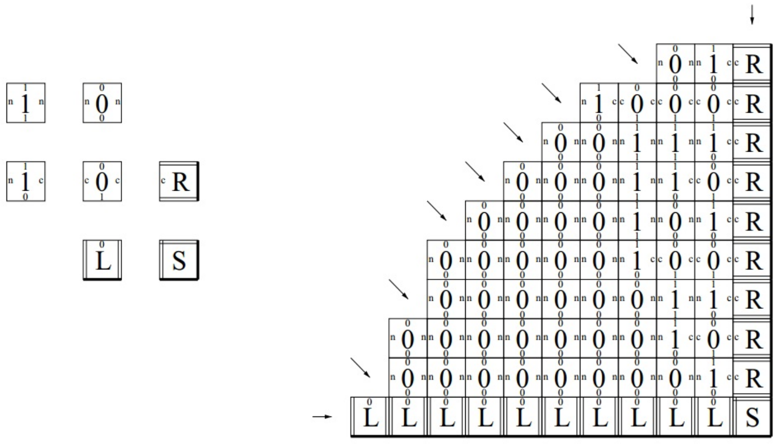


Figure 2.2: Simulating a binary counter with self-assembly. The set of tile types is depicted in the left side. In this figure, thick sides represent strength 0 glue, thin lines represent strength 1 glues and strength 2 glues are shown by double lined sides. The assembly shown in the right is not complete and the arrows indicates the positions that the assembly can grow further.

Combining the ideas in proof for Theorem 1 and Lemma 2 yields a better bound for assembling a full square at temperature 2.

Theorem 3. *[35] The tile complexity of an $N\times N$ full square at temperature 2 is at most $O(\log(N))$.*

Proof Sketch. We can reach this bound for assembling an $N\times N$ full square by combining the structures mentioned before. Let $n = \lceil \log(N)\rceil$ and $c = 1 + 2^{n-1} - \lceil N - n/2\rceil$. The

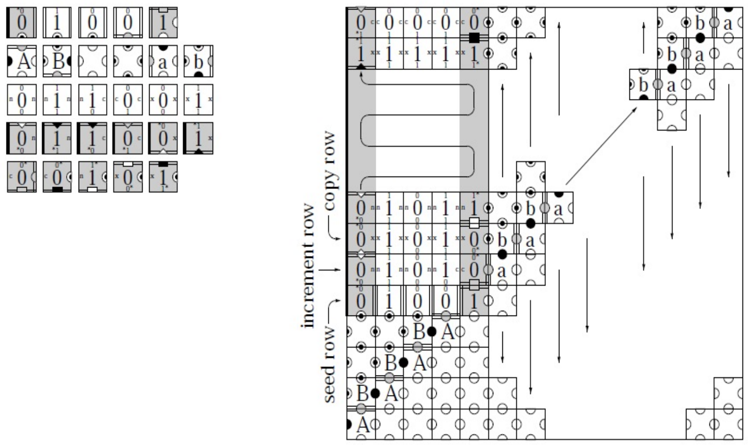


Figure 2.3: Constructing an $N \times N$ full square using $O(\log(N))$ tile types by combining structure in Figure 2.1 and Figure 2.2. Here $N = 52$, $n = 6$ and 28 tiles are used overall.

structure of the assembly is slightly different depending on the parity of the number $N - n$. The system shown in Figure 2.3 starts the assembly by constructing an $(n-1) \times (n-1)$ square with the method described in Theorem 1 with $n + 3$ tiles. But the seed row of this $(n-1) \times (n-1)$ square constructs the binary number $c - 1$ by encrypting each binary digit at each tile.

On the top of this smaller square the binary counter, that is similar to the one described in Lemma 2, starts counting from c to 2^{n-1}. The rows of this counter basically perform two main actions:

- *Increment rows:* As the name suggests, these rows increment the number on the row below themselves by 1. The glues n and c belong to the tiles constructing the *increment rows* in Figure 2.3. More precisely, c is used to simulate the action of carrying to the next digit and n indicates that action of carrying is completed. The assembly in these rows extends from right to left.

- *Copy rows:* These rows *copy* the binary number of the row below, by "reading" this number with the help of glues on the south side of their tiles. Conceptually, these tiles are responsible for reading the output of incrementing rows from the right side and fetch it to the next incrementing rows. The tiles that have glue x on their left or right side belong to *copy rows.* In these rows self-assembly grows from left to right.

Hence, two rows are used for each integer in the process of counting. Also, note that we use a special binding domain of strength 2 on the north side of the tiles corresponding to the first and last digits in each row. This is so that the system identifies the end of a row, and also to initiate the next row with the use of this binding domain of strength 2.
For the even $N - n$ the counter starts by (i.e. the seed row is) an increment row and it starts by a copy row for an odd $N - n$. When the number 2^{n-1} is reached the leftmost digit changes from 0 to 1. In this case there is a unique binding domain on the north side of the tile with the leftmost digit and the assembly of the counter halts by this tile. Also, there's a special tile for the rightmost digit of the first row after the seed row which initiates the assembly of the rest of the square diagonally with tiles labeled a and b in the picture. Overall, the counter has $O(n)$ tile types and the rest of the square is filled with 4 tiles on the diagonals and two blank tiles filling out the rest of the square as shown in Figure 2.3. $\square$

Most of the work that has been done on the tile complexity of self-assembly models is focused on temperature 2 assembly. While temperature 2 assembly yields very complex and efficient constructions, it comes with its own flaws. One of the main hurdles in experimental work on DNA self-assembly at temperature 2 is the high error rates in the order of attachments. Often tiles attach with less than strength 2 bond although the lab conditions are arranged to prevent this to happen [11]. From practical point of view, temperature 1 systems are more well-mannered and easier to control for laboratory experiments. For this reason, exploring the power of temperature 1 assembly can lead to influential improvements for experimental purposes.

Theorem 4. *[35] The tile complexity of an $N \times N$ full square at temperature 1 is N^2.*

Proof Sketch. Assume the seed tile is positioned at $(0, 0)$ in the $\mathbb{Z} \times \mathbb{Z}$ grid. To prove N^2 tile types suffice, we can simply place a unique tile at each position in the square with a unique binding domain between all adjacent pairs of tiles in the square. To prove necessity, assume there exists a tile assembly system $\mathbf{T}$ with tile set T, such that $|T| < N^2$ and $\mathbf{T}$ uniquely produces an $N \times N$ full square, A, as the terminal assembly. Since $|T| < N^2$, there exists some tile $r \in T$ that is present in two different positions, say $P_0 = (x_0, y_0)$ and $P_1 = (x_1, y_1)$, in A. Without loss of generality, we assume $x_0 < x_1$. Let $\phi(x, y)$ be the function translating P_0 to P_1 and let L be the shortest path between P_0 and P_1 in the backbone graph of A passing through (x_1, y_0) and let $Q = A[L]$.

Assume $R \in Asmb(\mathbf{T})$ is the assembly with fewest number of tiles in A, which connects

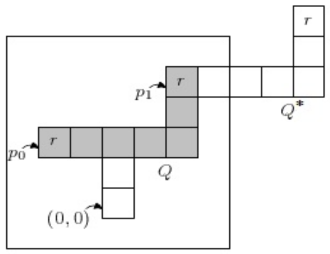


Figure 2.4: The assembly of $R \cup Q^* \in Asmb(\mathbf{T})$

the seed tile to Q, since A is a full square, R always exists. Since $R \in Asmb(\mathbf{T})$ there exists a $Q^* \in \{Q, \phi^{-1}(Q)\}$ such that $R \cup Q^* \in Asmb(\mathbf{T})$. More precisely, Q^* is Q if $x_1 > 0$ and Q^{-1} otherwise. This assembly can be extended indefinitely by adding more copies of translated versions of Q, i.e. $R \cup \bigcup_{i=1}^{n} \phi^n(Q^*) \in Asmb(\mathbf{T})$ is true for every integer $n \geq 0$ and this contradicts the assumption that the terminal assembly of $\mathbf{T}$ is a finite full square. Figure 2.4 depicts the idea of the proof. □

2.1.2 Tile Complexity of Full Squares in Other Models of Self-Assembly at Temperature 1

While, as seen before, the tile complexity of full squares at temperature 1, compared to temperature 2, is very large, it is shown, this is not necessarily the case for other models of tile-based self-assembly at temperature 1 [12, 29, 30].

Theorem 5. *[12] There exists a tile system under Staged tile assembly model that self-assembles an $N \times N$ full square at temperature 1 with $O(1)$ tiles, $O(1)$ bins and in $O(\log(N))$ stages.*

Proof Sketch. Basically, we start by decomposing an $N \times N$ hypothetical square, with no glues on the sides of its tiles, to fixed-size shapes in several levels with recursively using two simple operations on the shape, called *vertical decomposition* and *horizontal decomposition*. In a vertical decomposition a shape with n rows and m columns is divided into two shapes along the $i = \lfloor m + 1/2 \rfloor$th column such that all the tiles belonging to columns 1 (first) to $i - 1$ plus the two tiles on row 1 and n of the ith column, belong to one shape, which we call the *left* output of the operation. The rest of the tiles belong to the other shape, *right* output. The vertical decompositions continue until all the resulting shapes have at most 3 columns.

At this point these smaller shapes go through a series of horizontal decompositions, in which the shapes are divided by a straight cut into two shapes such that the difference between the number of rows in these two shapes is at most 1. These recursive decompositions continue until the number of rows in all the final shapes is at most 3 and by the end of the operations we end with little pieces of the starting square with at most 3 columns and 3 rows. This process can be shown with a rooted binary tree such that the $N \times N$ square shape is placed at the root and the children of each node are the outputs of either a vertical decomposition operation or a horizontal one. In a vertical decomposition, we define the left child of a node v to be associated with the left output of the vertical decompostion of the shape corresponding to v and the right child is defined as the shape corresponding to the right output of the vertical decomposition of v. Figure 2.5 shows the decomposition of an 8×8 square to constant-sized shapes.

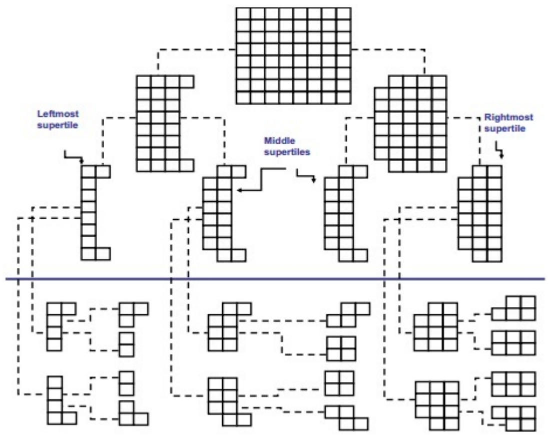


Figure 2.5: The binary tree depicting the process of decomposition of an 8×8 square to smaller shapes through the recursive use of vertical and horizontal decompositions [12].

Starting from the leaves of the tree, the glues are assigned to the boundaries of each shape (or nodes of the tree) in a bottom-up manner. For the nodes that have equal distance from the root (same *level* in the binary tree), same three colors are used to color those sides of the tiles that the decomposition cuts along them. Figure 2.6 illustrates this idea of assigning 3 colors (glues) to the boundaries. Ultimately, we use 3 sets, c_1, c_2 and c_3, of 3 distinct colors which we alternate between them level by level. The final pattern of coloring the boundaries of the shapes is shown in Figure 2.7. To see this, assign a distinct combination c_i to each letter a, b and c, in an arbitrary order, and use the same arrangement of these combinations on the boundaries of the shapes corresponding to leaves of the tree, based on their order in the tree (leftmost to rightmost) and recursively continue coloring border this way going up the tree. Note that, this color assignment is done first on the nodes that represent horizontal decompositions. As the coloring procedure goes up the tree, eventually the nodes representing vertical decompositions will be reached, in which point the color assignment will begin coloring the left and right boundaries.

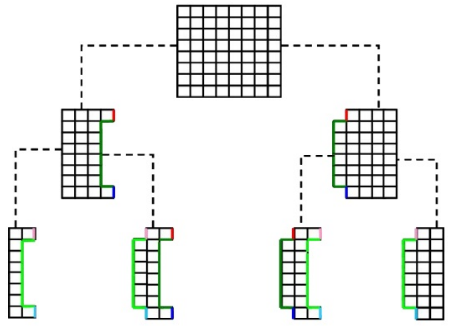


Figure 2.6: Assignment of constant number of colors to the shapes corresponding to the nodes of the binary tree [12].

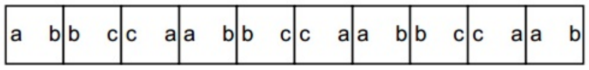


Figure 2.7: The figure illustrates the underlying idea in coloring the boundaries of the shapes that belong to the same level of the binary tree. Each letter in this figure corresponds to a combination of the 3 colors (c_i) and squares in the figure corresponds to shapes of the same level in the binary tree, such that the left most square corresponds to the left most shape (node) among the nodes of same level in the tree and also the right most square tile corresponds to the right most child. These combinations of 3 colors are assigned to the boundaries of the shapes as shown in Figure 2.6 [12].

Demaine *et al.* showed four bins suffice for the whole process of assembling a square with tiles that are designed in the manner described in Figure 2.6. The nodes in each level of the tree, are assembled in one stage, therefore the number of stages is dependent to the height of the tree which is of order $O(\log N)$. $\square$

Manuch *et al.* [29] proved that the assembly of a full squares under Step-wise assembly model can be done with a constant number of tile types as well.

Theorem 6. *[29] Under Step-wise assembly model, an $N \times N$ full square can be assembled* *at temperature 1 by using 4 distinct tile types if N is odd and 9 distinct tile types if N is even.*

Proof Sketch. First assume N is odd. The tile sequence $\{T_i\}_{i=1}^{N-1}$ in this case is defined as below:

$$T_i = \begin{cases} S_1 : \text{if } i \text{ is odd} \\ S_2 : \text{if } i \text{ is in form } 4k+2 \text{ for some } k \in N \\ S_3 : \text{if } i \text{ is divisible by } 4 \end{cases}$$

where S_1, S_2 and S_3 are depicted in Figure 2.8. The process of assembly of a 3×3 full square

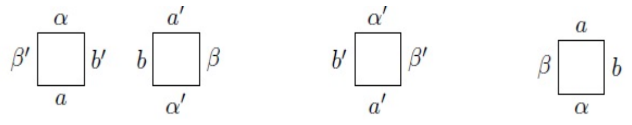


Figure 2.8: From left to right, the first two tile types belongs to S_1, the third tile type belongs to S_2 and the last one belongs to S_3 [29].

with this tile set sequence is shown in Figure 2.9. This process can be extended to even Ns

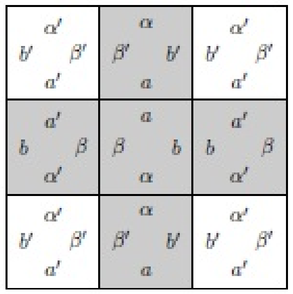


Figure 2.9: Tile sets S_1, S_2 and S_3 [29].

by adding some extra tile sets shown in Figure 2.10. The idea is to start by assembling an $(N-1)\times(N-1)$ square as shown above and attach the extra row and column by using the tile sets in Figure 2.10 in the order E_1, E_2 and E_3. The process of assembling a 4×4 square starting by a 3×3 square and using the tile sets E_is in Figure 2.10 is pictured in Figure 2.11.

□

At temperature 1, under Step-wise and Staged models a full square can be assembled much more efficiently than under aTAM. In Chapter 3 we prove that Step-wise and Staged

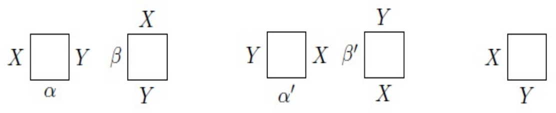


Figure 2.10: E_1 includes the first two tile types from left, E_2 includes the third and fourth tile type from the left and the rightmost tile type belongs to E_3 [29].

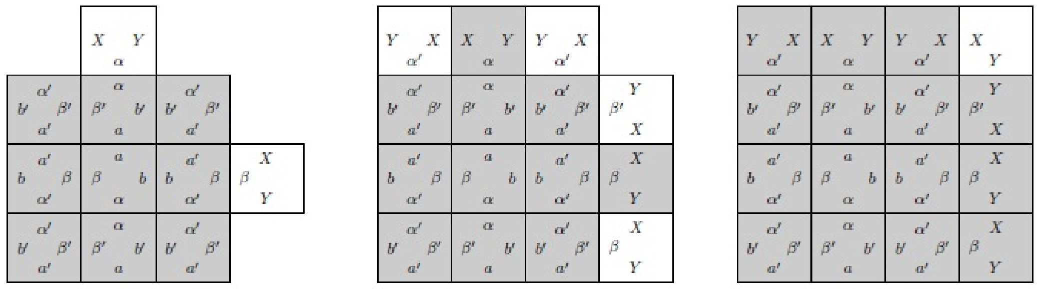


Figure 2.11: Assembling a 4×4 square by starting with a 3×3 square and in 3 extra steps with addtional tile sets E_1, E_2 and E_3 [29].

tile assembly models are in fact very powerful and are capable of simulating Universal Turing machine.

2.2 Turing Universality of Self-Assembly Models

The computational power of self-assembly models was exploited for the first time in [4], where linear self-assembly was used as a step in solving the Hamiltonian Path Problem. As mentioned in Chapter 1, the aTAM is computationally and geometrically powerful at temperature 2. At temperature 2, for the case where all the glues on a tile can form strength 1 bonds, the tile is not permitted to bond to the growing assembly until two tiles are already present on the neighboring locations to match the glues on the bonding sides. This cooperative behaviour grants a very beneficial control on the placement of new tiles and growth of the assembly. Winfree [43] proved that at temperature 2 the TAM is Turing Universal. This means, it is capable of simulating Universal Turing Machine, which is a Turing machine that can simulate (informally *act* as) an arbitrary Turing machine on an arbitrary input and produces the same result [22]. We refer the reader to [6] for further reading on description and existence of Universal Turing machine (UTM).

While at temperature 1 TAM is conjectured to be only capable of computing *simple* infinite shapes and patterns [15], but this is not the case for other models of tile assembly which are variants of aTAM. It is known that by adding simple constraints and features to the original model, universal computation is achievable [11, 30, 7]. In this section we review some of the results for two of these models, namely 3D tile assembly model and Restricted Glue tile assembly model (rgTAM). To this end, first we mention some important definitions. Note that, the rgTAM and 3D tile assembly model are described in Section 1.2.

Definition 7. *[11] A tile system $\Upsilon = \langle T', s', \tau' \rangle$* simulates a deterministic tile system $\Gamma = \langle T, s, \tau \rangle$ *at horizontal scale factor x and vertical scale factor y (or simply* simulates*), if there exists an onto function $SIM : R \mapsto T$, for some $R \subseteq T'$, such that for any non-empty tile type placed at position (i, j) in the terminal assembly of Γ, there exists a subset of tile types $R' \subseteq R$ where for each tile type $r \in R'$ such that $\Upsilon(k, l) = r$, for some k and l that $ix \leq k < (i+1)x$ and $jy \leq l < (j+1)y$ and $SIM(r) = \Gamma(i, j)$,*

for each position (i, j) such that $\Gamma(i, j) = empty$, then for each k and l such that $ix < k < (i+1)x$ and $jy < l < (j+1)y$, $\Upsilon(k, l) = empty$.

Informally, a system Υ simulates a system Γ if for each tile type t in system Γ a non-empty set of tile types, representing t, is placed by Υ at the same position but scaled up by some fixed vertical and horizontal scale. In other words, the simulating tile system should have a specific set of tile types which "acts" as each single tile types in Γ.

Definition 8. *[38] A* Turing machine *is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ and Γ are all finite and*

1. *Q is the set of states,*
2. *Σ is the input alphabet not containing **null**,*
3. *Γ is the tape alphabet where **null** $\in \Gamma$ and also $\Sigma \subseteq \Gamma$,*
4. *$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R, S\}$ is the transition function,*
5. *$q_0 \in Q$ is the start state,*
6. *q_{accept} is the accept state, and*
7. *q_{reject} is the reject state and $q_{accept} \neq q_{reject}$.*

Cook *et al* [11], developed an algorithm for transferring an arbitrary Turing machine to a Zig-Zag system with carefully constructing tiles.

Theorem 9. *[18] For a given Turing machine M with alphabet Σ and alphabet $w \in \Sigma^*$ there exists a temperature 2 Zig-Zag tile system that simulates M over input w with tile complexity polynomial in number of states of M and size of w. More precisely, there exists a Zig-Zag system that, given a seed assembly consisting of a horizontal line with north glues denoting an input string, the assembly will place a unique accept tile type if and only if T accepts the input string, and a unique reject tile type if and only if M rejects the input string.*

They used Theorem 9 to prove Turing universality for 3D tile assembly model at temperature 1.

Theorem 10. *[11] For any 2 dimensional temperature 2 Zig-Zag tile system $\Gamma = \langle T, s, 2\rangle$, there exists a 3D temperature 1 tile system that simulates Γ at vertical scale $= 4$, horizonal scale $O(\log |T|)$, and tile complexity $O(|T| \log(|T|))$.*

Proof Sketch. Here we present the basic idea behind the proof of Theorem 10. The goal is to design a 3D tile system $\Gamma' = \langle T', s', 1\rangle$ which simulates Γ. To this end, for each tile type in T a set of tile types is added to T' that assembles a configuration, called a *macro-tile*, which will "fake" the cooperative attachment of tiles in temperature 2 with geometrical enforcements. In this process we simulate any east/west bonds or any strength 2 bonds in Γ with a single strength 1 glue in Γ'.

Let G be the set of all strength 1 glues on the north and south sides of the tiles that appeared in the system Γ. Label each type of these glues with numbers 0 to $|G| - 1$ in an arbitrary order. For any glue $g \in G$ let $b(g)$ be the binary representation of the label of g. For a copy tile $t \in T$, which has input glues x on its east side and input glue y on its south side, the assembly of the macro-tile starts when the macro-tiles corresponding to the east and south nieghbors of t are fully assembled. Assume the assembly of this macro-tile starts in plane $z = i$. The assembly of this macro-tile starts by a glue $(x, -)$, this glue knows the input glue on the east is x but has no information about the glue on the south of t. The glue on the south is *read* in its binary form bit by bit. Each bit, 0 or 1, has a specific assembly path. For example for the kth ($k = 0, \dots, \lceil \log(|G| - 1) \rceil$) bit, there exists a specific set of tile types in T' which can read 0 and another set of tiles in T' which can read 1. Each of these tile sets assembles a unique path and either of them can attach to the assembly of the macro-tile which has by now read glue x and the first $k-1$ bits of $b(y)$. To ensure that the macro-tile reads the correct number as the kth bit, one of these paths (which corresponds to the wrong bit) is blocked by the pre-built assemblies which are already placed in the necessary position. These assemblies are placed while assembling the macro-tile corresponding to the tile which is present at the south side of t in Γ, say t'. Let m' be the macro-tile corresponding to t'. The tile set T' is designed such that, during the assembly process of m', these *blocking tiles* are guaranteed to be placed in the correct positions in $z = i$ based on the north glue of t' which is y. After all the bits of $b(y)$ are read, then the assembly process continues by propagating the output glues which are on the west and north side of t. There exists a specific set of tiles in T' for this task. As before, the north glue of t is assembled bit by bit in the macro-tile and while writing each bit, a new set of blocking tiles are placed in the correct positions in $z = i+1$ which are going to block the assembly path of the macro-tile of the tile above t to make sure it reads the correct bits of the glue on its south (which is the north glue of t). A similar idea can be used for simulating tiles with input glues on their west and south sides.

In a nutshell, the assembly process of the tiles that are in row i of the Zig-Zag assembly is being simulated in plane $z = i$ and is controlled by the blocking tiles that are already placed in $z = i$ (these tiles are placed when the tiles of the row $i-1$ were being simulated in Γ'). And while simulating the tiles in row i of the Zig-Zag assembly, the next blocking tiles are placed in $z = i+1$ based on the binary representation of glues on the north side of the tiles in row i. These blocking tiles in $z = i+1$, control the assembly path which reads the glues on the south side of the macro-tiles corresponding to the tiles in row $i+1$. Assume m is the macro-tile corresponding to some copies of the tile type t, which has input glues c and d on its west (or east) and south side, respectively. m first "reads" c and $b(d)$ and then by assembling the blocking tiles to make sure that in future the macro-tiles that correspond to the tiles that has north glue of these copies of t as their south input, read this glue correctly.

Thus, there exists a subset of T' that uniquely produces a macro-tile for a tile type t with input glues c and d. Each tile $t \in T$ can have up to two distinct sets of input glues. To be exact, it has two sets of input glues if and only if a copy of t has its west side glue as its input glue in a position and another copy of t has its east side glue as its input at another position in the assembly. Each macro-tile has $O(\log(|G|))$ distinct tile types and for each tile in T there is at most two macro-tiles, which leads to $O(|T| \log |T|)$ tile types in T'.

□

Next we review a similar result for rgTAM.

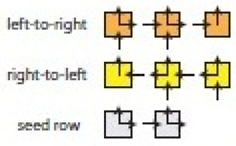
Theorem 11. *[30] For every zig-zag Tile Assembly System $\boldsymbol{T} = \langle T, s, 2\rangle$, there exists a restricted Tile Assembly Model $\boldsymbol{S} = \langle S, s', 1\rangle$ such that $\boldsymbol{S}$ simulates $\boldsymbol{T}$ at horizontal scale factor factor $O(\log(|T|))$ and $|S| = O(|T|\log(|T|))$.*

Proof Sketch. Again, we map each tile type in T to a set of tiles in T' that assembles a supertile called *macro-tile*. A tile t can have an input glue either on east side, west side or neither, but never both. A macro-tile must be self-assembled in whole before self-assembly of another $macro-tile$ starts. Let G be the number of unique strength 1 north/south glues in T. In this construction, each macro-tile contains $2\lceil \log(G)\rceil + 30$ tiles.

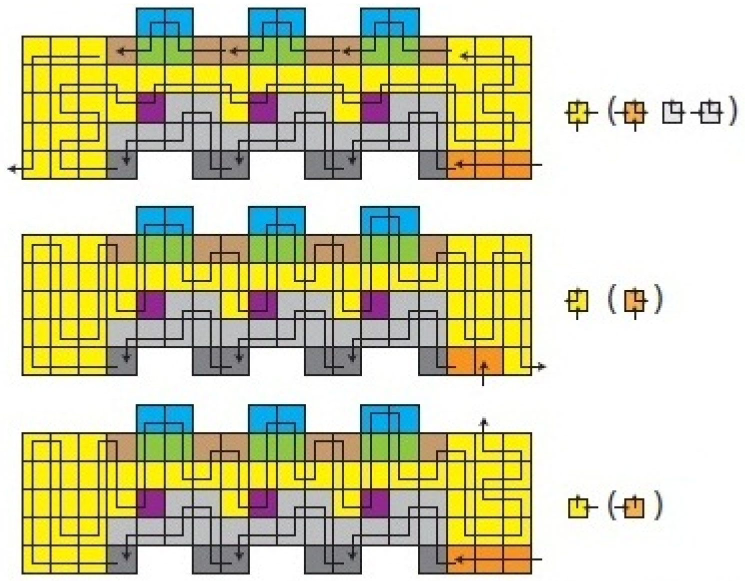
We label each strength 1 north/south glue in T with a unique $\lceil \log(|G|)\rceil$-bit binary number. We geometrically represent each of these binary numbers with bumps and dents on specific positions along the north/south side of each macro-tile. Each dent and bump will uniquely represent one binary digit based on the location of the negative glue on specific sides of it. The west/east glues and glues of strength 2 are encoded with a strength 1 glue on the beginning or end of each macro-tile. We briefly sketch the construction procedure for a sample macro-tile simulating a tile type with input glues on south and east. The macro-tile that emulates other tile types are constructed in a similar manner. The self-assembly of the macro tile contains two main logical steps: reading the input glues and then writing the output glues. The assembly of the macro-tile starts by binding to the macro-tile on its east side via the glue on the east and then the tiles that assembles the south side of each macro-tile read the binary number, from the north side of the macro-tile beneath, bit by bit as the assembly grows. The placement of negative glue determines if the bit is 0 or 1 as shown in Figure 2.12(c). The tile types that read a binary string from the top of an existing macro-tile as they self-assemble from right to left are depicted in Figure2.13. Once all bits have been read, the output glues are determined since $\mathbf{T}$ is detereministic. As the assembly goes on along the north side of the macro-tile beneath, the bits of the glue on the south are read bit by bit until all the bits are read. An example depicting the north side of a macro-tile being read by the south side of another macro-tile is shown in Figure 2.14.

After the output glues of the macro-tile have been determined, the macro-tile path crawls back across itself and starts writing the north output glue. This is accomplished in a manner that is similar to reading a $\lceil \log(|G|) \rceil$-bit binary string. Again, we use $O(\log(|G|))$ distinct tile types to do this. The tile types for this task are shown in Figure 2.15.

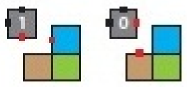
$\square$



(a) The set of input and output side combinations in a Zig-Zag assembly system



(b) Macro-tiles simulate the behaviour of tiles in Γ. The arrow shows the direction of the growth and the schematic tiles on the right side pictures the related input and output.



(c) Reading 1 bit of the south bond.

Figure 2.12: Creating macro-tiles according to possible input and output sides of copies of tiles in **T**.

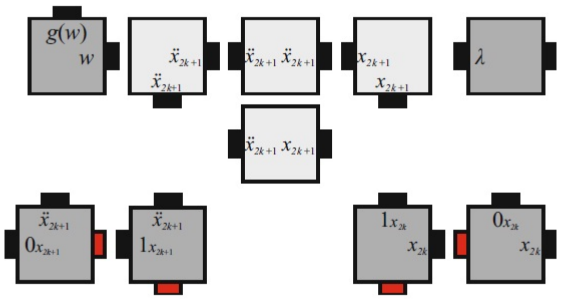


Figure 2.13: The tile types that read a binary string from the top of an existing macro-tile as they self-assemble from right to left. For each $i = 0, 1, \ldots, G + 1$, let $x_i = \{0, 1\}^i$. Note that the east input glue is implicitly encoded into the tile types shown here. The upper right tile initiates the process of reading the input binary string. The upper left tile completes the process by mapping a pair of south-east input glues to the corresponding north-west output glues. Tiles for south-west input macro tiles are designed similarly.

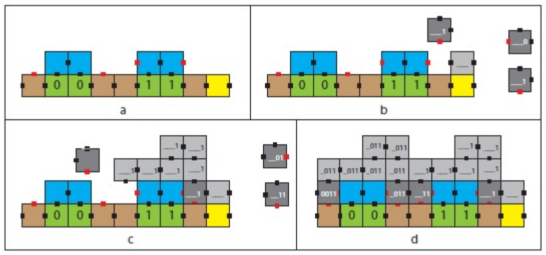


Figure 2.14: An example of a macro-tile "reading" the glue from the north side of the macro-tile beneath. The glue is represented by binary number "0011". The assembly grows from right to left and it reads the binary number bit by bit as the assembly grows. The negative glue (pictured in red) determines if the bit that is being read is 0 or 1, by restricting the tiles that can be place at each dent.

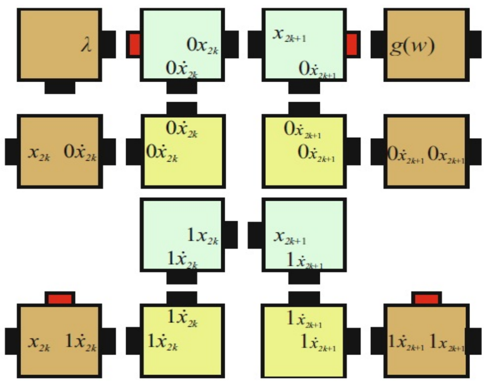


Figure 2.15: The tile types responsible for writing the output glue on the north side of a macro tiles. For each $i = 0, 1, \ldots, \lceil \log(|G|) \rceil$, let $x_i = \{0, 1\}^i$.

Chapter 3

Turing Universality Results

In this chapter, we present our results on computational power of two tile self-assembly models, namely Step-wise tile self-assembly model and Staged tile self-assembly model, at temperature 1. As we mentioned in Section 2.1.2, at temperature 1, assembly of an $N \times N$ full square under these two models requires only a constant number of tile types. On the other hand the tile complexity of a full square is proved to be N^2 under the aTAM (see Theorem 4). This suggests that these models might be capable of assembling complex shapes more efficiently than the aTAM. Here, we prove that both these models are capable of simulating an arbitrary Turing machine over an arbitrary input.

3.1 Turing Universality of Step-wise Tile Assembly Model at Temperature 1

In this section, we present our result on Step-wise tile assembly model. We design a Step-wise tile assembly system for simulating an arbitrary Zig-Zag tile assembly system. Then, according to Theorem 9, it is possible to simulate an arbitrary Turing machine in Step-wise model by simulating the corresponding Zig-Zag system.

Theorem 12. *[7] For any deterministic temperature 2 Zig-Zag tile assembly system* $\Gamma = \langle \Sigma, T, s, s_\Sigma, 2 \rangle$ *there exists a Step-wise tile assembly system* $\Upsilon = \langle \Sigma', T', s', s'_{\Sigma'}, 1 \rangle$ *at temperature 1 that simulates* Γ *at vertical scale* $y = 5$*, horizontal scale* $x = O(\log(|T|))$ *and with tile complexity* $O(|T| \log(|T|))$*.*

According to the definition of simulation (see Definition 7), we have to design a system Υ

such that there exists an onto function $\mathrm{SIM} : R \mapsto T$ for some $R \subseteq T'$, such that for any tile $\Gamma(i, j)$ at position (i, j) in the terminal assembly of Γ, there must exist a unique non-empty tile type $t' \in R$, such that $\Gamma'(k, l) = t'$ for some $k = \log(|\Sigma|)$ and l where $ix \leq k < (i + 1)x$ and $jy \leq l < (j + 1)y$ and $\mathrm{SIM}(t') = \Gamma(i, j)$.

To this end, first we prove the following lemma and in part Final Steps of the Proof we present the proof for Theorem 12.

Lemma 13. *For any deterministic temperature 2 Zig-Zag tile assembly system* $\Gamma = \langle \Sigma, T, s, s_\Sigma, 2\rangle$ *there exists a Step-wise tile assembly system* $\Gamma' = \langle \Sigma', T', s', s'_{\Sigma'}, 1\rangle$ *at temperature 1, for which there is a bijective function* $f : R' \mapsto T$*, where* $R' \subseteq \mathcal{P}(T')$ *and* $\mathcal{P}(T')$ *is the set of all subsets of* T'*, such that for each* $(i, j) \in \mathrm{Struc}(\mathrm{Term}(\Gamma))$ *where* $\Gamma(i, j) = t \neq$ *empty, there exists a unique subset* $T^* \in R'$ *with a tile* $t' \in T^*$*, such that* $\Gamma'(k, l) = t'$ *for some* $k = \log(|NS|)$ *and* l*, where* $ix \leq k < (i + 1)x$ *and* $jy \leq l < (j + 1)y$*, and* $f(T^*) = t$ *and* $t' \notin \bigcup_{\forall t'' \in T, t'' \neq t} f^{-1}(t'')$.

Proof. The proof is constructive. We design a system Γ' that satisfies the conditions of this lemma. In our construction for Γ', we simulate the growth of assemblies in Γ at temperature 2 in Γ' at temperature 1. Intuitively, Γ' simulates Γ by converting each tile type $t \in T$ into a group of tile types in Γ' that self-assemble into a *macro-tile*. A "macro-tile" is a supertile, which is in shape close to a rectangle. As mentioned in Section 1.1, a copy of a tile type $t \in T$ attaches to the growing structure via its input sides and it interacts with its succeeding tiles in the tile sequence via the output sides. The macro-tiles corresponding to this copy of t "fakes" in Γ' this functionality of t in Γ. This macro-tile "reads" the inputs of t with help of geometrical enforcements (bumps and dents) that prevent erroneous growth of assembly and then maps the read inputs to the corresponding outputs. This is done by some specific tiles in the system called *mapping tiles.* Once the outputs are determined, the macro-tile "writes" the outputs by specific geometrical representations. According to properties of Zig-Zag assembly model at temperature 2, a tile in $\mathrm{Term}(\Gamma)$ might have up to 2 inputs. For the case that the number of inputs is 2, the macro-tile must read two input glues. To make this action possible at temperature 1, one glue is transferred (read) via a single strength 1 bond in the macro-tile and the other glue is read by geometrical enforcements. In this way, macro-tiles can simulate the cooperative behavior of temperature 2 systems at temperature 1.

Our construction for Γ' is similar to what is presented in [30, 11], except that we improve the tile complexity of the final system from $O(|T|\log(|T|))$ in [30] to $O(\min\{|T|, |\Sigma|^2\}\log(|T|))$ by introducing the new idea of mapping tiles which allows us to reuse input/output interfaces of macro-tiles and therefore possibly saving in number of tile types. In what follows, we present this construction for Γ' in detail and will describe the process of geometrical simulation in macro-tiles. First, we define some notations that we will be using throughout the proof.

Let $\{terminal\} = \mathrm{Term}(\Gamma)$. For a tile type $t \in T$, the function $\mathrm{IO}(t)$ returns the set of all feasible input/output side combinations for t in Γ. The function IO will be computed by Algorithms 1 and 2.

For every $(i, j) \in \mathrm{Struc}(terminal)$, define

$$\mathrm{Input}(i,j) = \{d : d \in D, \Gamma(i,j)_d \text{ is an input glue of } \Gamma(i,j) \text{ at } (i,j) \text{ via side } d\},$$

and

$$\mathrm{Output}(i,j) = \{d : d \in D, \Gamma(i,j)_d \text{ is an output glue of } \Gamma(i,j) \text{ at } (i,j) \text{ via side } d\}.$$

For all $t \in T$, define

- $pos(t) = \{(i,j) : (i,j) \in \mathrm{Struc}(terminal), \Gamma(i,j) = t\}$ is the set of all position where tile type t appears in *terminal*,
- $\mathrm{I}(t) = \{I : (I,O) \in \mathrm{IO}(t)\}$ and $\mathrm{O}(t) = \{O : (I,O) \in \mathrm{IO}(t)\}$ are the sets of possible input (output) glues for tile type t,
- for all $(I,O) \in \mathrm{IO}(t)$, let $\mathrm{Cop}(t,(I,O)) = \{(i,j) \in pos(t) : \mathrm{Input}(i,j) = I, \mathrm{Output}(i,j) = O\}$ be the set of positions containing tile type t with input glues I and output glues O.

We have the following simple observation. For every tile type $t \in T$, $\{(\mathrm{Input}(i,j), \mathrm{Output}(i,j) : (i,j) \in pos(t)\} \subseteq \mathrm{IO}(t)$.

Some Observations on Assemblies under Zig-Zag systems at Temperature 2 In this part, we review some properties of *terminal*, using the notations defined above.

Observation 14. *For all $t \in T$ and $(I,O) \in \mathrm{IO}(t)$, $|I| \leq 2$.*

Observation 15. *For all $t \in T$ and $(I,O) \in \mathrm{IO}(t)$, if $I = \{d\}$ for some $d \in D$ (i.e. $|I| = 1$), then $s_\Sigma(t_d, t_d) = 2$.*

Algorithm 1 The algorithm for computing IO(t).

```
**Input**: a tile type $t \in T$
**Output**: IO($t$)
$IO(t) = \emptyset$
if $t = s$ then
  $in = \emptyset$
  $out =$OUTPUT$(W, N)$
  Add $(I, O)$ to IO($t$)
end if
if $s_\Sigma(t_s, t_s) = 2$ then
  $in = \{S\}$
  $O_1 =$OUTPUT$(E, N)$
  $O_2 =$OUTPUT$(W, N)$
  $out = O_2 \cup O_1$
  Add $(I, O)$ to IO($t$)
end if
if $s_\Sigma(t_E, t_E) = 2$ then
  $in = \{E\}$
  $out =$OUTPUT$(W, N)$
  Add $(I, O)$ to IO($t$)
end if
if $s_\Sigma(t_W, t_W) = 2$ then
  $in = \{W\}$
  $out =$OUTPUT$(E, N)$
  Add $(I, O)$ to IO($t$)
end if
if $s_\Sigma(t_S, t_S) = 1$ and $s_\Sigma(t_E, t_E) = 1$ then
  $in = \{E, S\}$
  $out =$OUTPUT$(W, N)$
  Add $(I, O)$ to IO($t$)
end if
if $s_\Sigma(t_S, t_S) = 1$ and $s_\Sigma(t_W, t_W) = 1$ then
  $in = \{W, S\}$
  $out =$OUTPUT$(E, N)$
  Add $(I, O)$ to IO($t$)
end if
```

Algorithm 2 OUTPUT(t, d_1, d_2): returns the possible output sides of t among d_1, d_2

```
**Input**: tile $t$, directions $d_1, d_2$
**Output**: OUTPUT$(t, d_1, d_2)$
OUTPUT$(t, d_1, d_2) = \emptyset$
**if** $t_{d_1} \neq \_$ **then**
  add $d_1$ to OUTPUT$(t, d_1, d_2)$
**end if**
**if** $t_{d_2} \neq \_$ **then**
  add $d_2$ to OUTPUT$(t, d_1, d_2)$
**end if**
```

If a tile is not positioned at the beginning or end of any row in *terminal*, then we call it an *internal* copy, otherwise we call it a *boundary* copy.

Observation 16. *For all $t \in T$ and $(I, O) \in \mathrm{IO}(t)$, if $E \in I$ and $W \in O$, or vice versa, then by properties of the Zig-Zag model, the tiles at positions in* $\mathrm{Cop}(t, (I, O))$ *are all internal copies. In addition, $\{E, W\} \subseteq I \cup O$ if and only if all the tiles placed at positions in* $\mathrm{Cop}(t, (I, O))$ *are internal copies.*

Observation 17. *For all $t \in T$ and $(I, O) \in \mathrm{IO}(t)$, if either $I = \{d\}$ or $O = \{N\}$, where $d = \in \{N, S\}$, then by the properties of the Zig-Zag model, the tiles at positions in* $\mathrm{Cop}(t, (I, O))$ *are all boundary copies. In addition, $I = \{d\}$ or $O = \{N\}$, if and only if all the tiles placed at the positions in* $\mathrm{Cop}(t, (I, O))$ *are boundary copies. Moreover, according to Observation 14, the north or south bond corresponding to side d must have strength 2.*

Note that the last part of the above observation is consistent with the fact that in the Zig-Zag assembly model at temperature 2, the assembly can enter a row only with a strength 2 bond.

Next, we define function Label which will be mapping every north (south) side of every tile type $t \in T$ which is not in $\mathrm{O}(t)$ $(\mathrm{I}(t))$ to a special glue g^*, and mapping all other sides to the original glues. This will be used later to prevent mismatches in macro-tiles. For every $t \in T$ and $(I, O) \in IO(t)$, the function

$$\mathrm{Label} : \bigcup_{t \in T} \{N, S\} \times \{t\} \times \mathrm{IO}(t) \mapsto G,$$

where $G = \Sigma \cup \{g^*\}$, is defined as follows:

$$\text{Label}(N, t, (I, O)) = \begin{cases} t_N, & \text{if } N \in O \\ g^*, & \text{if } N \notin O \end{cases} \tag{3.1}$$

$$\text{Label}(S, t, (I, O)) = \begin{cases} t_S, & \text{if } S \in I \\ g^*, & \text{if } S \notin I \end{cases} \tag{3.2}$$

Then, we define $NS = \bigcup_{d \in \{S,N\}, t \in T, (I,O) \in \text{IO}(t)} \{\text{Label}(d, t, (I, O))\}$.

Since NS is a subset of $\Sigma \cup \{g^*\}$, it is obvious that $|NS| = O(|\Sigma|)$. Assign to each $g \in NS \setminus \{g^*\}$ a unique number from 1 to $|NS|$, and let b_g be the binary representation of the number assigned to g. For g^*, let $b_{g^*} = \epsilon$, the empty string.

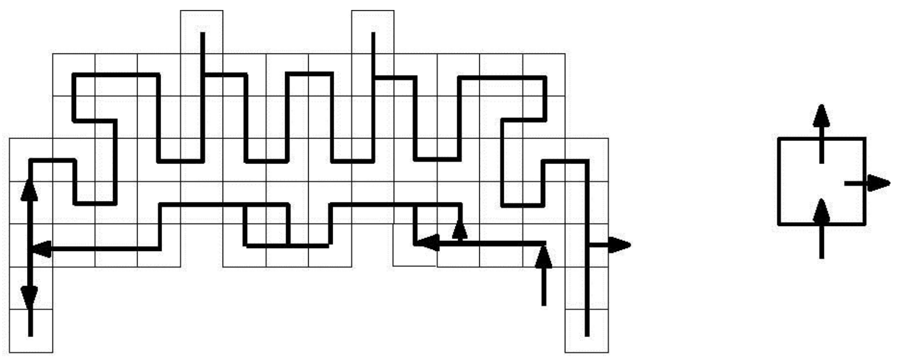
Macro-tiles and Geometrical Encoding In this part, we describe the simulation process. We use the idea of simulation by macro-tiles from [30]. A "macro-tile" is a supertile, which is in shape close to a rectangle. We describe the macro-tiles in details later in part Classification and Algorithms.

For each tile type $t \in T$ we design $|\text{IO}(t)|$ macro-tile types. We design function Macro : $\bigcup_{t \in T} \{t\} \times \text{IO}(t) \mapsto \mathcal{P}(T')$ which will map each $(I, O) \in \text{IO}(t)$ to the set of tiles types that assemble together to create the specific macro-tile type for tile type t with input/output side combination (I, O). The function Macro is computed by Algorithms 4, 6 and 8 which are stated in part Classifications and Algorithms. Later in part Final Steps of the Proof, we define bijection f using the function Macro which will satisfy the conditions of the lemma.

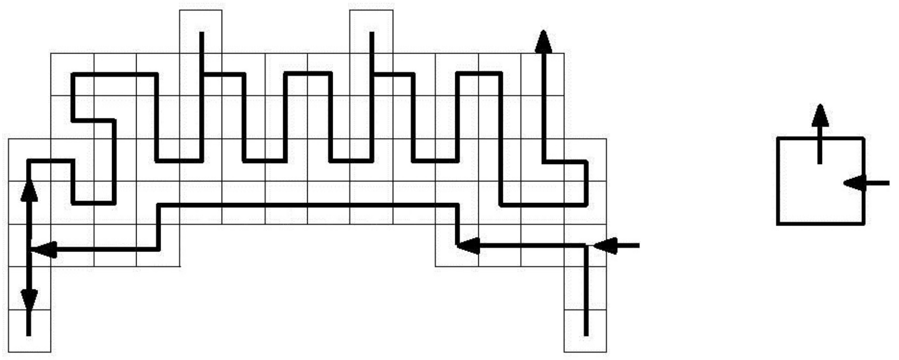
The tile types of each macro-tile type will be split into two sets, namely T'_1 and T'_2. These two sets will be used to design the sequence of sets of tile types, $T' = \{T_n\}_{n \geq 1}$ which is the sequence of sets of tile types of Γ'. The sequence will alternate between two sets of tile types T'_1 and T'_2. More formally, T_n is defined as follows,

$$T_n = \begin{cases} T'_1, & \text{if } n \text{ is odd.} \\ T'_2, & \text{if } n \text{ is even.} \end{cases} \tag{3.3}$$

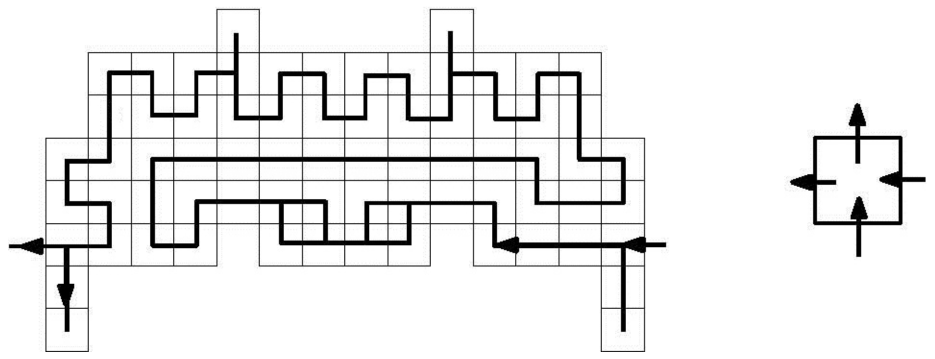
Each macro-tile has four boundaries (see Figure 3.1). We denote the boundary on side d of a macro-tile m by m_d, where $d \in D$. The macro-tile Macro$(t, (I, O))$ "reads" the input glues via the boundaries corresponding to the directions in set I and "writes" the output glues on the boundaries corresponding to the directions in set O.



(a) A macro-tile corresponding to a tile with input glue via south, and output glues via east and north.



(b) A macro-tile corresponding to a tile with input glue via east, and output glue via north.



(c) A macro-tile corresponding to a tile with input glues via east and south, and output glues via west and north.

Figure 3.1: In Γ′, macro-tiles simulate the behavior of tiles in Γ. In each subfigure the macro-tile on the left simulates a tile of Γ on the right together with its particular input/output side combination. In macro-tiles the arrows show the direction of growth, and in the tiles (on the right) the arrows represent the input and output sides.

For a macro-tile $m = \text{Macro}(t, (I, O))$, if $N \in O$ then the binary representation of t_N, b_{t_N}, is geometrically encoded on the north boundary of m by a sequence of "bumps", where each bump encodes one digit of b_{t_N}, see Figure 3.1. If $S \in I$, the south boundary of m geometrically decodes b_{t_S} as described in the next paragraph. If there is a side $d \in N \cup S$ that $d \notin I \cup O$, then m_d contains no bumps, as illustrated in Figure 3.1(b) for $d = S$. In particular, the geometrical representation of $g \in NS \setminus \{g^*\}$ is based on the digits of b_g, where each bit, 0 or 1, is represented by the position (right or left) of the bump on the boundary, we call these positions 0 and 1, respectively. Each bump is a tile of the boundary with three exposed sides. Figure 3.2 illustrates the geometric representation of each bit 0 and 1. To read and write each bit, the simulation will use a constant number of tiles in T_1' and T_2'.

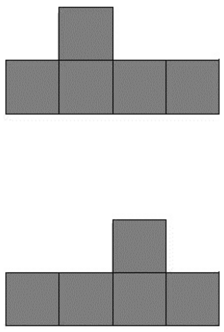


Figure 3.2: The geometrical representation for bits 0 and 1 for an east growing row. The top figure pictures a bump at position 1. The bottom figure pictures a bump at position 0.

Next, we describe how the north boundary of a macro-tile enforces the south boundary of the macro-tile above it. Assume $g \in NS\backslash\{g^*\}$ is assigned to the south side of a copy of tile type t in *terminal* located at position (i, j), i.e., $g = \text{Label}(S, t, (\text{Input}(t, (i, j)), \text{Output}(t, (i, j)))$. The south boundary of macro-tile $\text{Macro}(t, (\text{Input}(t, (i, j)), \text{Output}(t, (i, j))))$ reads b_g bit by bit by immersing the structure first in T_1' and then in T_2' as follows. To read each bit from the north side of the macro-tile corresponding to $\Gamma(i, j-1)$, the construction of macro-tiles will guarantee that structure continues to grow in two branches: east and north. If the bump is at position 1, the east branch is blocked by this bump and north branch continues to grow when the current structure is immersed in T_2', and if the bump is at position 0, then the east branch continues to grow in the set T_1' and blocks the north branch before we immerse the structure in T_2'. When the structure finishes to grow in T_2' both branches finish at the same position and the glue on the exposed side of the last tile placed contains the information of the bits read so far. For details see Figure 3.2. Once all the bits are read the exposed glue on the last tile has the information of all input glues. Since Γ is deterministic, this information uniquely determines the tile in the system Γ which is being simulated, and hence the output glues are determined as well. The tile of the macro-tile which attaches next, can be assigned a glue which contains the unique information about the output glues of the macro-tile. We call this tile a *mapping tile*. The purpose of the mapping tile is to initiate the assembly for the rest of the macro-tile with the correct output glues. Note that mapping tiles will be crucial for defining bijection f. In the next section, we describe the location of mapping tiles within the macro-tiles explicitly, as well as the glues on the sides of tiles in Γ so that tiles of Γ' can form all the required macro-tiles.

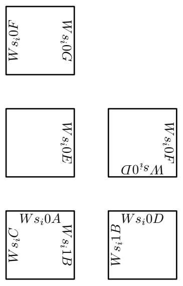
Notice that the south boundary of each macro-tile has two *vertical teeth*, one attached to the south side of the tile that assembly of the macro-tile starts with and one attached to the south side of the last tile on south boundary of the macro-tile. Although these vertical teeth are not necessary for the construction we present for Step-wise assembly, but in Section 3.2, they will be crucial for the system presented for Staged tile assembly model.

Classification and Algorithms. In this part, we present the algorithms for constructing T', which generate the tile types in T'_1 and T'_2 for assembling each macro-tile. For each tile type $t \in T$ and each $(I, O) \in \mathrm{IO}(t)$ these algorithms describe the tile types in T' that assemble the corresponding macro-tile.

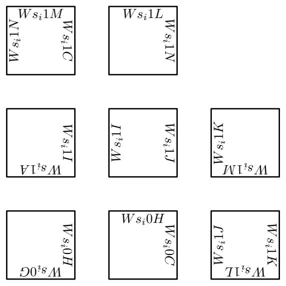
To that end, we classify the elements of $\mathrm{IO}(T)$ to 5 classes. The construction for each class is slightly different, although the idea remains the same. Algorithm 3 classifies the input/output side combinations to the desired 5 classes.

Consider a tile type $t \in T$ with input and output side combination $(I, O) \in \mathrm{IO}(t)$ and assume $S \in I$. The construction of tile types in T' responsible for reading the south input is the same for all the classes and is done Algorithm 4, by adding appropriate tile types into T'_1 and T'_2. We define the glues in Σ' as triples (x, y, z) where $x \in \Sigma$, y is a binary string and $z \in Z$ determines the function of tile within its macro-tile. For any binary string b let $b^1b^2\ldots b^{|b|}$ be the sequence digits of b. Let $k = \lceil \log_2 |NS| \rceil$. The Algorithm 4 constructs the tiles using the pattern for reading bits shown in Figure 3.1 repeatedly.

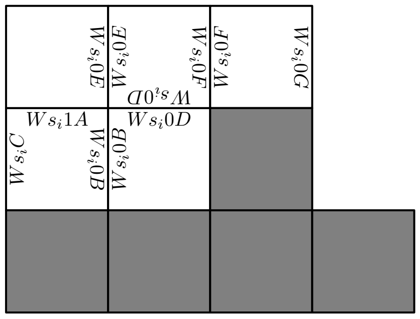
Assume $d \notin I \cup O$, where $d \in \{N, S\}$ and let $m = \mathrm{Macro}(t, (I, O))$. In this case, if $d = N$, the assembly does not write any output on the north side of m, and if $d = S$, the



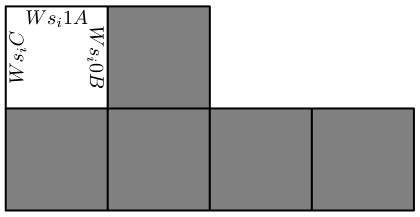
(a) The subset of T'_1 reading the ith bit from left to right.



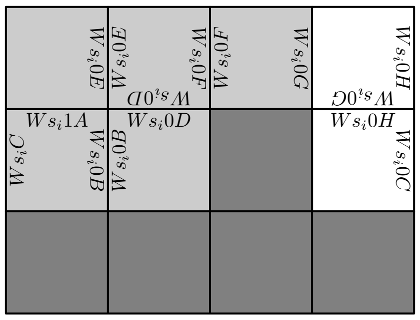
(b) The subset of T'_2 reading the ith bit from left to right.



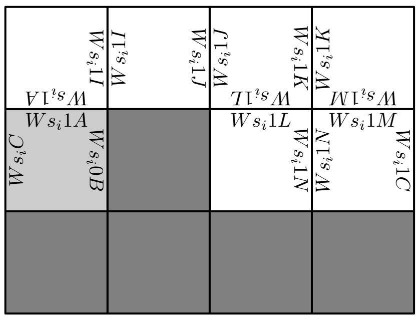
(c) The final assembly reading bit 0 after exposing to the first tile set. The tiles that are colored gray belong to the macro-tile beneath.



(d) The final assembly reading bit 1 after exposing to the first tile set. The tiles that are colored gray belong to the macro-tile beneath.



(e) The final assembly reading bit 0. The that are colored light gray tiles belong to T'_1.



(f) The final assembly reading bit 0. The tiles that are colored light gray belong to T'_1.

Figure 3.3: The tiles and final assemblies for reading the ith bit of b_{t_S} for cases that Label$(S, t, (I, O)) \neq g^*$. The tiles necessary for reading either 0 or 1 is included in T' for each bit, but the already placed tiles of the macro-tile below guarantees the system assembles only the right assembly. In the figures the dark gray tiles belong to the macro-tile below. W in the glue labels represent t_W and s_i represent the bits read up to this point. The letter $A \dots N$ determines the location of tiles inside the macro-tile.

Algorithm 3 Classify the input/output side combinations of tiles in T to 5 classes.

```
Input: $t \in T$ and $(I, O) \in \mathrm{IO}(t)$
Output: $\mathrm{classify}(t, (I, O)) \in \{1, 2, 3, 4, 5\}$
if $I = \emptyset$ then
  $\mathrm{classify}(t, (I, O)) = 4$
else if $O = \emptyset$ then
  $\mathrm{classify}(t, (I, O)) = 5$
else if $\{E, W\} \subseteq I \cup O$ and $I \neq \{S\}$ then
  $\mathrm{classify}(t, (I, O)) = 1$
else if $I = \{S\}$ then
  $\mathrm{classify}(t, (I, O)) = 2$
else if $O = \{N\}$ then
  $\mathrm{classify}(t, (I, O)) = 3$
end if
```

macro-tile does not read any input from the south. Therefore, there are no bumps or dents on m_d. For example, for the case $d = S$, we assemble the south boundary as a horizontal one layer sequence of tiles of length $4k - 2$, which passes the input (from west or east) along (see Figure 3.1(b)). This enables this macro-tile to be compatible with any macro-tile below it (possibly creating mismatches). Algorithms 6 and 7 generates the tiles types to assemble this sequence. The same idea is used to assemble the north boundary for the case $d = N$. Algorithms 8 and 9 generates the tile types to assemble the north boundary in this case. Both algorithms can be applied to all 5 classes if such d exists.

In what follows, we study each class according to Algorithm 3, respectively, and for each class we describe the direction of growth and the mapping tile in the macro-tiles corresponding to a $t \in T$ and $(I, O) \in \mathrm{IO}(t)$.

Class 1 ($\mathrm{classify}(t, (I, O)) = 1$**).** Recall in this class both east and west glues are either an input or an output for t and $I \neq \{S\}$. According to Observation 16 this copy of t is an internal copy. Figure 3.4 illustrates a macro-tile corresponding to t and (I, O) in class 1 which belongs to an east growing row of the Zig-Zag assembly (input via the west side). In this figure the tile has input glues via both south and west sides and output glues via north and east sides. For the opposite direction see Figure 3.1(c).

The assembly of macro-tile starts by reading the west glue, and reads the south input as it grows on south boundary. When both inputs are read the mapping tile attaches to the

Algorithm 4 The algorithm generates the tile types in Macro$(t,(I,O))$, for reading the south input, when $I=\{S,W\}$.

```
Input: $k = \log(|NS|)$, $t \in T$ and $(I,O) \in \mathrm{IO}(t)$ such that $I = \{S,W\}$
Output: Corresponding tile types in $T'_1$ and $T'_2$
$g = \mathrm{Label}\,(S,t,(I,O))$
if $W \in I$ then
  for $i = 0$ to $k-1$ do
    Add the following tile types to $T'_1$:
    $[\,(t_W, b_g^0 \dots b_g^{i-1}1, A)\,,\,(t_W, b_g^0 \dots b_g^{i-1}0, B)\,,\,\_\,,\,(t_W, b_g^0 \dots b_g^{i-1}0, C)\,]$
    $[\,(t_W, b_g^0 \dots b_g^{i-1}0, D)\,,\,\_\,,\,\_\,,\,(t_W, b_g^0 \dots b_g^{i-1}0, B)\,]$
    $[\,\_\,,\,(t_W, b_g^0 \dots b_g^{i-1}0, E)\,,\,\_\,,\,\_\,]$
    $[\,\_\,,\,(t_W, b_g^0 \dots b_g^{i-1}0, F)\,,\,(t_W, b_g^0 \dots b_g^{i-1}0, D)\,,\,(t_W, b_g^0 \dots b_g^{i-1}0, E)\,]$
    $[\,\_\,,\,(t_W, b_g^0 \dots b_g^{i-1}0, G)\,,\,\_\,,\,(t_W, b_g^0 \dots b_g^{i-1}0, F)\,]$
    Add the following tile types to $T'_2$:
    $[\,\_\,,\,\_\,,\,(t_W, b_g^0 \dots b_g^{i-1}0, H)\,,\,(t_W, b_g^0 \dots b_g^{i-1}0, G)\,]$
    $[\,(t_E, b_g^0 \dots b_g^{i-1}0, H)\,,\,(t_W, b_g^0 \dots b_g^{i-1}0, C)\,,\,\_\,,\,\_\,]$
    $[\,\_\,,\,(t_W, b_g^0 \dots b_g^{i-1}1, I)\,,\,(t_W, b_g^0 \dots b_g^{i-1}1, A)\,,\,\_\,]$
    $[\,\_\,,\,(t_W, b_g^0 \dots b_g^{i-1}1, J)\,,\,\_\,,\,(t_W, b_g^0 \dots b_g^{i-1}1, I)\,]$
    $[\,\_\,,\,(t_W, b_g^0 \dots b_g^{i-1}1, K)\,,\,(t_W, b_g^0 \dots b_g^{i-1}1, L)\,,\,(t_W, b_g^0 \dots b_g^{i-1}1, J)\,]$
    $[\,(t_W, b_g^0 \dots b_g^{i-1}1, L)\,,\,(t_W, b_g^0 \dots b_g^{i-1}1, N)\,,\,\_\,,\,\_\,]$
    $[\,\_\,,\,\_\,,\,(t_W, b_g^0 \dots b_g^{i-1}1, M)\,,\,(t_W, b_g^0 \dots b_g^{i-1}1, K)\,]$
    $[\,(t_W, b_g^0 \dots b_g^{i-1}1, M)\,,\,(t_W, b_g^0 \dots b_g^{i-1}1, C)\,,\,\_\,,\,(t_W, b_g^0 \dots b_g^{i-1}1, N)\,]$
  end for
end if
```

Algorithm 5 The algorithm generates the tile types in Macro$(t,(I,O))$, for reading the south input, when $I=\{S,E\}$.

```
Input: $k = \log(|NS|)$, $t \in T$ and $(I,O) \in \mathrm{IO}(t)$ such that $I = \{S,E\}$
Output: Corresponding tile types in $T_1'$ and $T_2'$
$g = \mathrm{Label}\,(S,t,(I,O))$
if $E \in I$ then
  for $i = 0$ to $k-1$ do
    Add the following tile types to $T_1'$:
    $[\,(t_E, b_g^0 \dots b_g^{i-1}0, A)\,,\,(t_E, b_g^0 \dots b_g^{i-1}, C)\,,\,\_\,,\,(t_E, b_g^0 \dots b_g^{i-1}1, B)\,]$
    $[\,(t_E, b_g^0 \dots b_g^{i-1}1, D)\,,\,(t_E, b_g^0 \dots b_g^{i-1}1, B)\,,\,\_\,,\,\_\,]$
    $[\,\_\,,\,\_\,,\,\_\,,\,(t_E, b_g^0 \dots b_g^{i-1}0, E)\,]$
    $[\,\_\,,\,(t_E, b_g^0 \dots b_g^{i-1}0, F)\,,\,(t_E, b_g^0 \dots b_g^{i-1}1, D)\,,\,(t_E, b_g^0 \dots b_g^{i-1}0, E)\,]$
    $[\,\_\,,\,(t_E, b_g^0 \dots b_g^{i-1}0, F)\,,\,\_\,,\,(t_E, b_g^0 \dots b_g^{i-1}0, G)\,]$
    Add the following tile types to $T_2'$:
    $[\,\_\,,\,(t_E, b_g^0 \dots b_g^{i-1}0, G)\,,\,(t_E, b_g^0 \dots b_g^{i-1}0, H)\,,\,\_\,]$
    $[\,(t_E, b_g^0 \dots b_g^{i-1}0, H)\,,\,\_\,,\,\_\,,\,(t_E, b_g^0 \dots b_g^{i-1}0, C)\,]$
    $[\,\_\,,\,\_\,,\,(t_E, b_g^0 \dots b_g^{i-1}1, A)\,,\,(t_E, b_g^0 \dots b_g^{i-1}1, I)\,]$
    $[\,\_\,,\,(t_E, b_g^0 \dots b_g^{i-1}1, I)\,,\,\_\,,\,(t_E, b_g^0 \dots b_g^{i-1}1, J)\,]$
    $[\,\_\,,\,(t_E, b_g^0 \dots b_g^{i-1}1, J)\,,\,(t_E, b_g^0 \dots b_g^{i-1}1, L)\,,\,(t_W, b_g^0 \dots b_g^{i-1}1, K)\,]$
    $[\,(t_E, b_g^0 \dots b_g^{i-1}1, L)\,,\,\_\,,\,\_\,,\,(t_E, b_g^0 \dots b_g^{i-1}1, N)\,]$
    $[\,\_\,,\,(t_E, b_g^0 \dots b_g^{i-1}1, K)\,,\,(t_E, b_g^0 \dots b_g^{i-1}1, M)\,,\,\_\,]$
    $[\,(t_E, b_g^0 \dots b_g^{i-1}1, M)\,,\,(t_E, b_g^0 \dots b_g^{i-1}1, N)\,,\,\_\,,\,(t_E, b_g^0 \dots b_g^{i-1}1, C)\,]$
  end for
end if
```

Algorithm 6 The algorithm generates the tile types in Macro$(t, (I, O))$, for reading the south input, when $I = \{E\}$ ($S \notin I$).

```
Input: $k = \log(|NS|)$, $t \in T$ and $(I, O) \in \mathrm{IO}(t)$ such that $I = \{E\}$
Output: Corresponding tile types in $T_1'$ and $T_2'$
if Label$(S, t, (I, O)) = g^*$ and $E \in I$ then
  Add the following tile types to $T_1'$
  $[\,(t_E, \epsilon, S)\,,\,(t_E, \epsilon, S)\,,\,\_\,,\,\_\,]$
  $[\,\_\,,\,\_\,,\,(t_E, \epsilon, S)\,,\,(t_E, \epsilon, A_0)\,]$
  $[\,\_\,,\,(t_E, \epsilon, B_0)\,,\,\_\,,\,(t_E, \epsilon, C_0)\,]$
  Add the following tile types to $T_2'$
  $[\,\_\,,\,(t_E, \epsilon, A_0)\,,\,\_\,,\,(t_E, \epsilon, B_0)\,]$
  $[\,\_\,,\,(t_E, \epsilon, C_0)\,,\,\_\,,\,(t_E, \epsilon, D_0)\,]$
  for $i = 1$ to $k - 2$: do
    Add following tile types to $T_1'$:
    $[\,\_\,,\,(t_E, \epsilon, D_{i-1})\,,\,\_\,,\,(t_E, \epsilon, A_i)\,]$
    $[\,\_\,,\,(t_E, \epsilon, B_i)\,,\,\_\,,\,(t_E, \epsilon, C_i)\,]$
    Add the following tile types to $T_2'$:
    $[\,\_\,,\,(t_E, \epsilon, A_i)\,,\,\_\,,\,(t_E, \epsilon, B_i)\,]$
    $[\,\_\,,\,(t_E, \epsilon, C_i)\,,\,\_\,,\,(t_E, \epsilon, D_i)\,]$
  end for
  Add following tile types to $T_1'$:
  $[\,\_\,,\,(t_E, \epsilon, D_{k-2})\,,\,\_\,,\,(t_E, \epsilon, A_{k-1})\,]$
  $[\,\_\,,\,(t_E, \epsilon, B_{k-1})\,,\,\_\,,\,(t_W, \epsilon, C_{k-1})\,]$
  Add the following tile types to $T_2'$:
  $[\,\_\,,\,(t_E, \epsilon, A_{k-1})\,,\,\_\,,\,(t_E, \epsilon, B_{k-1})\,]$
  $[\,\_\,,\,(t_E, \epsilon, C_{k-1})\,,\,(t_E, \epsilon, D_{k-1})\,,\,\_\,]$
  $[\,(t_E, \epsilon, D_k)\,,\,\_\,,\,\_\,,\,(T_E, b_{t_N}, P)\,]$
end if
```

Algorithm 7 The algorithm generates the tile types in Macro$(t, (I, O))$, for reading the south input, when $I = \{W\}$ $(S \notin I)$.

```
Input: $k = \log(|NS|)$, $t \in T$ and $(I, O) \in \mathrm{IO}(t)$ such that $I = \{W\}$
Output: Corresponding tile types in $T_1'$ and $T_2'$
if Label$(S, t, (I, O)) = g^*$ and $W \in I$ then
  Add the following tile types to $T_1'$
  $[\,(t_W, \epsilon, S)\,,\ \_\ ,\ \_\ ,\ (t_W, \epsilon, S)\,]$
  $[\,\_\ ,\ (t_W, \epsilon, S)\,,\ \_\ ,\ (t_W, \epsilon, A_0)\,]$
  $[\,\_\ ,\ (t_W, \epsilon, B_0)\,,\ \_\ ,\ (t_W, \epsilon, C_0)\,]$
  Add the following tile types to $T_2'$
  $[\,\_\ ,\ (t_W, \epsilon, B_0)\,,\ \_\ ,\ (t_W, \epsilon, A_0)\,]$
  $[\,\_\ ,\ (t_W, \epsilon, D_0)\,,\ \_\ ,\ (t_W, \epsilon, C_0)\,]$
  for $i = 1$ to $k - 2$: do
    Add following tile types to $T_1'$:
    $[\,\_\ ,\ (t_W, \epsilon, A_i)\,,\ (t_W, \epsilon, D_i)\,,\ \_\ ])$
    $[\,\_\ ,\ (t_W, \epsilon, C_i)\,,\ \_\ ,\ (t_W, \epsilon, B_i)\,]$
    Add the following tile types to $T_2'$:
    $[\,\_\ ,\ (t_W, \epsilon, B_i)\,,\ \_\ ,\ (t_W, \epsilon, A_i)\,]$
    $[\,\_\ ,\ (t_W, \epsilon, D_i)\,,\ \_\ ,\ (t_W, \epsilon, C_i)\,]$
  end for
  Add following tile types to $T_1'$:
  $[\,\_\ ,\ (t_W, \epsilon, A_{k-1})\,,\ (t_W, \epsilon, D_{k-1})\,,\ \_\ ]$
  $[\,\_\ ,\ (t_W, C_{k-1})\,,\ \_\ ,\ (t_W, \epsilon, B_{k-1})\,]$
  Add the following tile types to $T_2'$:
  $[\,\_\ ,\ (t_W, \epsilon, B_{k-1})\,,\ \_\ ,\ (t_W, \epsilon, A_{k-1})\,]$
  $[\,\_\ ,\ \_\ ,\ (t_W \epsilon D_k)\,,\ (t_W, \epsilon, C_{k-1})\,]$
  $[\,(t_W, \epsilon, D_k)\,,\ (T_E, b_{t_N}, P)\,,\ \_\ ,\ \_\ ]$
end if
```

Algorithm 8 The algorithm generates the tile types in Macro$(t, (I, O))$, for assembling the north boundary, when $O = \{E\}$ $(N \notin O)$.

```
Input: $k = \log(|NS|)$, $t \in T$ and $(I, O) \in \mathrm{IO}(t)$ such that $N = \{E\}$
Output: Corresponding tile types in $T_1'$ and $T_2'$
if Label$(N, I, O, t) = g^*$ and $E \in I$ then
   Add the following tile types to $T_1'$
   $[\_ , (t_E, \epsilon, N) , (t_W, \epsilon, N) , \_ ]$
   $[ (t_W, \epsilon, N) , \_ , \_ , (t_E, \epsilon, A_0) ]$
   $[\_ , (t_W, \epsilon, B_0) , \_ , (t_W, \epsilon, C_0) ]$
   Add the following tile types to $T_2'$
   $[\_ , (t_W, \epsilon, A_0) , \_ , (t_W, \epsilon, b_g^0) ]$
   $[\_ , (t_W, \epsilon, C_0) , \_ , t_W, \epsilon, D_0 ])$
   for $i = 1$ to $k - 1$: do
      Add following tile types to $T_1'$:
      $[\_ , (t_W, \epsilon, D_{i-1}) , \_ , (t_W, \epsilon, A_i) ]$
      $[\_ , (t_W, \epsilon, B_i) , \_ , (t_W, \epsilon, C_i) ]$
      Add the following tile types to $T_2'$:
      $[\_ , (t_W, \epsilon, A_i) , \_ , (t_W, \epsilon, B_i) ]$
      $[\_ , (t_W, , \epsilon, C_i) , \_ , (t_W, \epsilon, D_i) ]$
   end for
   Add following tile types to $T_1'$:
   $[\_ , (t_W, \epsilon, D_{k-1}) , \_ , (t_W, \epsilon, A_{k-1}) ]$
   $[\_ , (t_W, \epsilon, B_{k-1}) , \_ , (t_W, \epsilon, C_{k-1}) ]$
   Add the following tile types to $T_2'$:
   $[\_ , (t_W, \epsilon, A_{k-1}) , \_ , (t_W, \epsilon, k - 1) ]$
   $[\_ , (t_W, \epsilon, C_{k-1}) , (t_W, \epsilon, D_k) , \_ ]$
   $[ (t_W, \epsilon, D_k) , \_ , \_ , (T_W, , \epsilon, F) ]$
end if
```

Algorithm 9 The algorithm generates the tile types in Macro$(t, (I, O))$, for assembling the north boundary, when $O = \{W\}$ $(N \notin O)$.

```
Input: $k = \log(|NS|)$, $t \in T$ and $(I, O) \in \text{IO}(t)$ such that $N = \{W\}$
Output: Corresponding tile types in $T'_1$ and $T'_2$
if Label$(N, I, O, t) = g^*$ and $W \in I$ then
  Add the following tile types to $T'_1$
  $[\,\_\,,\ \_\,,\ (t_E, \epsilon, S)\,,\ (t_E, \epsilon, S)\,]$
  $[\,\_\,,\ (t_E, \epsilon, S)\,,\ \_\,,\ (t_E, \epsilon, A_0)\,]$
  $[\,\_\,,\ (t_E, \epsilon, b_g^0)\,,\ \_\,,\ (t_E, \epsilon, C_0)\,]$
  Add the following tile types to $T'_2$
  $[\,\_\,,\ (t_E, \epsilon, b_g^0)\,,\ \_\,,\ (t_E, \epsilon, A_0)\,]$
  $[\,\_\,,\ (t_E, \epsilon, D_0)\,,\ \_\,,\ (t_E, \epsilon, C_0)\,]$
  for $i = 1$ to $k - 2$: do
    Add following tile types to $T'_1$:
    $[\,\_\,,\ (t_E, \epsilon, A_i)\,,\ (t_E, \epsilon, D_i)\,,\ \_\,]$
    $[\,\_\,,\ (t_E, \epsilon, C_i)\,,\ \_\,,\ (t_E, \epsilon, B_i)\,]$
    Add the following tile types to $T'_2$:
    $[\,\_\,,\ (t_E, \epsilon, B_i)\,,\ \_\,,\ (t_E, \epsilon, A_i)\,]$
    $[\,\_\,,\ (t_E, \epsilon, D_i)\,,\ \_\,,\ (t_E, \epsilon, C_i)\,]$
  end for
  Add following tile types to $T'_1$:
  $[\,\_\,,\ (t_E, \epsilon, A_{k-1})\,,\ (t_E, \epsilon, D_{k-1})\,,\ \_\,]$
  $[\,\_\,,\ (t_E, \epsilon, C_{k-1})\,,\ \_\,,\ (t_E, \epsilon, B_{k-1})\,]$
  Add the following tile types to $T'_2$:
  $[\,\_\,,\ (t_E, \epsilon, B_{k-1})\,,\ \_\,,\ (t_E, \epsilon, A_{k-1})\,]$
  $[\,\_\,,\ \_\,,\ (t_E, \epsilon, D_k)\,,\ (t_E, \epsilon, C_{k-1})\,]$
  $[\,\_\,,\ (T_E, \epsilon, F)\,,\ (t_E, \epsilon, D_k)\,,\ \_\,]$
end if
```

assembly. If $W \in I$, the mapping tile is of the form,

$$[\ (t_E, b_{t_N}, P)\ ,\ \_\ ,\ \_\ ,\ (t_W, b_{t_S}, P)\ ],$$

and if $E \in I$, the mapping tile is,

$$[\ (t_W, b_{t_N}, P)\ ,\ (t_E, b_{t_S}, P)\ ,\ \_\ ,\ \_\ ].$$

When the mapping tile attaches to the assembly, the growing process switches its direction toward west by some *filler* tiles. The filler tiles are the tiles in the assembly that are not involved in the process of reading inputs or writing outputs, but rather pass the information to the required positions in the macro-tile for writing and reading inputs and outputs, respectively. If $N \in O$, when the assembly process reaches the north boundary, it starts writing the the north glue, t_N, of the original tile $t \in T$ by placing bumps in position 0 or 1 on the north boundary according to the digits of b_{t_N}. In the following we describe this process in detail. Recall that for reading the south input the assembly process had grown two branches one for each possible bit. Of these two branches only one survived in the assembly and the other one was blocked by a bump. Unlike this approach, for writing the north output, the assembly process will grow only one branch for each bit in b_{t_N}. Note that all the tile types for assembling the north boundary are in tile set T_1'. We can do this since the assembly process proceeds only along one branch. Let $g' = \text{Label}\,(N, t, (I, O))$. Algorithm 10 adds appropriate tile types to T_1' for assembling the north boundary of each macro-tile $\text{Macro}(t, (I, O))$ such that $N \in O$.

If $N \notin O$, we use Algorithm 8 and 9 to generate the tile types for assembling the north boundary. Once the north boundary is assembled, the assembly process continues growing on the east or west boundary depending on whether $E \in O$ or $W \in O$. For example, in Figure 3.4 it is east boundary. If $E \in O$, the last tile which creates the east output with a single bond on the east boundary is $[\ \_\ ,\ (t_E, \epsilon, S)\ ,\ (t_E, \epsilon, V)\ ,\ (t_E, \epsilon, Q)\ ]$, and if $W \in O$, this tile is $[\ \_\ ,\ (t_W, \epsilon, Q)\ ,\ (t_W, \epsilon, V)\ ,\ (t_W, \epsilon, S)\ ]$.

Class 2 (classify$(t, (I, O)) = 2$**).** Recall that in this class $I = \{S\}$, and all the other sides with non-null glues are included in the output set. According to Observation 17, $s_\Sigma(t_S, t_S) = 2$ and the copies of t are boundary copies. Since the only input is via the south side, these copies must be the first tile in the rows they belong to in *terminal*, in other words, they are the tiles that enter the next row. The assembly process of the macro-tiles for this class starts with the mapping tile. The mapping tile for t and (I, O) in this class is

Algorithm 10 The algorithm generates the tile types in Macro$(t,(I,O))$, for writing the north output on the north boundary, when $N \in O$.

```
Input: $k = \log(|NS|)$, $t \in T$ and $(I,O) \in \mathrm{IO}(t)$ such that $N \in O$
Output: Corresponding tile types in $T_1'$
$g' = \mathrm{Label}\,(N, t, (I,O))$
if $W \in O$ then
  for $i = 0$ to $k-1$ do
    if $b_i = 0$ then
      Add the following tile types to $T_1'$:
      $[\,\_\,,\,\_\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}0, A')\,,\,(t_W, b_{g'}^0 \dots b_{g}^{i-1}0, B')\,]$
      $[\,(t_W, b_{g'}^0 \dots b_{g}^{i-1}0, B')\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}0, C')\,,\,\_\,,\,\_\,]$
      $[\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}0, D')\,,\,\_\,,\,\_\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}0, C')\,]$
      $[\,(t_W, b_{g'}^0 \dots b_{g}^{i-1}0, E')\,,\,(t_W, b_{g'}^0 \dots b_{g}^{i-1}, F')\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}0, D')\,,\,\_\,]$
      $[\,\_\,,\,\_\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}, G')\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}, F')\,]$
      $[\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}, G')\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}, H')\,,\,\_\,,\,\_\,]$
      $[\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}, I')\,,\,\_\,,\,\_\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}, H')\,]$
      $[\,\_\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}, A')\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}, I')\,,\,\_\,]$
    end if
    if $b_i = 1$ then
      Add the following tile types to $T_1'$:
      $[\,\_\,,\,\_\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}1, A')\,,\,(t_W, b_{g'}^0 \dots b_{g}^{i-1}1, B')\,]$
      $[\,(t_W, b_{g'}^0 \dots b_{g}^{i-1}1, B')\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}1, C')\,,\,\_\,,\,\_\,]$
      $[\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}1, D')\,,\,\_\,,\,\_\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}1, C')\,]$
      $[\,\_\,,\,(t_W, b_{g'}^0 \dots b_{g}^{i-1}1, E')\,,\,(t_W, b_{g'}^0 \dots b_{g}^{i-1}1, D')\,,\,\_\,]$
      $[\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}1, F')\,,\,\_\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}, G')\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}1, E')\,]$
      $[\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}, G')\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}, H')\,,\,\_\,,\,\_\,]$
      $[\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}, I')\,,\,\_\,,\,\_\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}, H')\,]$
      $[\,\_\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}, A')\,,\,(t_W, b_{g'}^0 \dots b_{g'}^{i-1}, I')\,,\,\_\,,\,]$
    end if
  end for
end if
```

[$\_$, (t_E, ϵ, P) , $(\_ , b_{t_S}, S)$, (t_W, b_{t_N}, P)], regardless of the parity of the row (either east growing or west growing). The reading tiles that assemble the south boundary are produced by Algorithms 4 and 5.

Next we describe the assembly process for the filler tiles and the north boundary. If $s_\Sigma(t_N, t_N) = 2$, the process of assembly is illustrated in Figure 3.5. The filler and writing tiles are designed such that the direction of growth follows the arrows pictured in Figure 3.5. This design is the same for the both cases when t is in an even or odd row.

If $s_\Sigma(t_N, t_N) < 2$, the only difference from the previous case is that there is no glue on the north boundary. Figure 3.6 illustrates the direction of growth for this case, regardless of the parity of the row.

Note that the assembly process for macro-tiles in this class may not finish before the assembly process of the next macro-tile starts. But it is guaranteed that the assembly process for these macro-tiles finishes before the assembly process of the succeeding macro-tile.

Also notice that if both $E, W \in O$ then both glues will be written on east and west boundaries, but we will show that before the assembly process for the next macro-tile finishes the assembly process continues to grow in the correct direction. Assume a copy of t with input/output side combination (I, O) is placed at location (i, j) in *terminal* and $d \in \{E, W\}$ is the direction of growth for the row below j and therefore the wrong direction for row j. According to the properties of Zig-Zag systems, $S(d(i, j))$ has to be empty in *terminal*. Thus, when reading the south input there are no bumps to block the 0 bit branch and the south boundary will read the all 0 binary sequence. Recall that number 0 is not assigned to any $g \in NS$ (i.e. there is no $g \in NS$ with $b_g = 0$). Therefore there is no mapping tile, and hence macro-tile, with inputs t_d and 0, and the assembly for this direction terminates when $k = \log(|NS|)$ bits of 0 are assembled.

Class 3 (classify$(t, (I, O)) = 3$**).** Recall that in this case $O = \{N\}$ but $I \neq \{S\}$. The copies of t are all boundary copies and since the only output is via the north side, we can deduce that these copies are the last tiles in their rows and since the row of the copy of t is not the last row in the *terminal*, $s_\Sigma(t_N, t_N) = 2$. Both directions of growth in this case are illustrated in Figure 3.7. If $E \in I$ then the mapping tile type is,

$$[\ \_\ ,\ (t_E, b_{t_S}, P)\ ,\ \_\ ,\ (\_ , b_{t_N}, P)\ ],$$

and if $W \in I$, the mapping tile type is of the form,

$$[\ (\_ , b_{t_N}, P)\ ,\ \_\ ,\ \_\ ,\ (t_E, b_{t_S}, P)\ ].$$

Class 4 (classify$(t, (I, O)) = 4$**).** Recall that in this class $I = \emptyset$ which implies that this copy of t is the seed tile in Γ (i.e. $t = s$). Hence, the macro-tile $m = \text{Macro}(t, (I, O))$ is the first macro-tile that is assembled in Γ' and the seed tile of Γ' is the tile that starts the assembly process for m. The assembly process of a macro-tile m in this class starts with its mapping tile, which is $s' = [\,\_\,, (t_E, b_{t_N}, P)\,, \_\,, \_\,]$. Algorithm 6and 7 generates the corresponding tile types in T'_1 and T'_2 for assembling the south boundary and Algorithm 10 generates the tile types for assembling north boundary if $N \in O$, otherwise if $N \notin O$ the tile types for assembling the north boundary are generated by Algorithm 8. As before, the glue t_W (the glue on the west side of t) is transferred to the neighboring macro-tile via a strength 1 glue on the west boundary of m.

Class 5 (classify$(t, (I, O)) = 5$**).** In this class $O = \emptyset$ and $I \neq \emptyset$. Therefore, the assembly of *terminal* ends with a copy of t. The input boundaries are assembled as before by tile types generated by Algorithms 4 and 6. The tile type for assembling the north boundary is constructed with Algorithm 8.

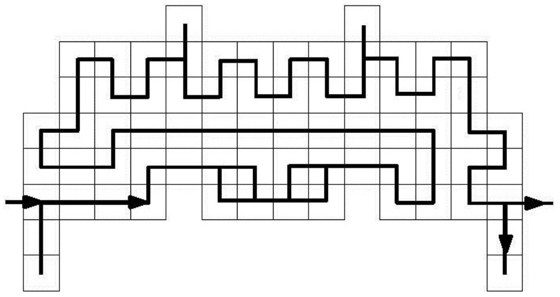


Figure 3.4: A macro-tile corresponding to a tile with input glues via west and south sides, and output glues via east and north sides.

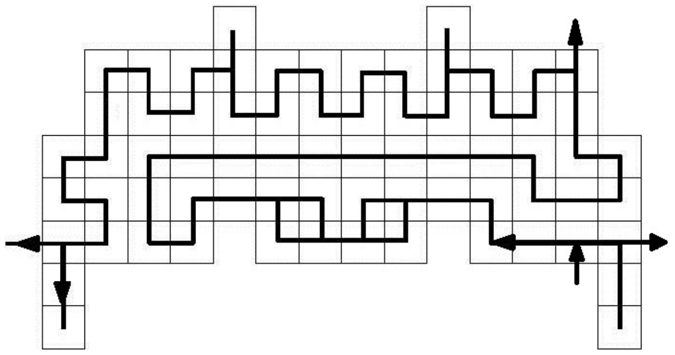


Figure 3.5: General shape of a macro-tile, for a tile types $t \in T$ and $(I, O) \in \mathrm{IO}(t)$, with $I = \{S\}$ and $s_\Sigma(t_N, t_N) = 2$.

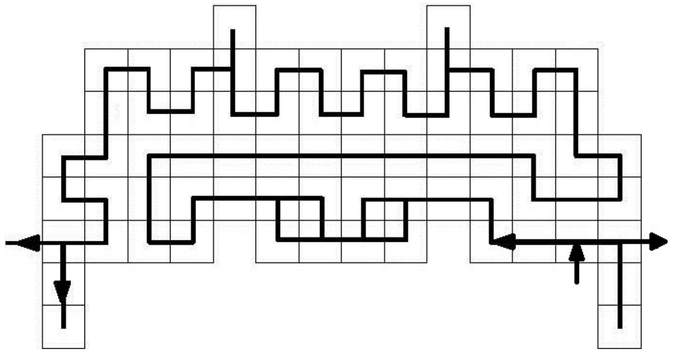
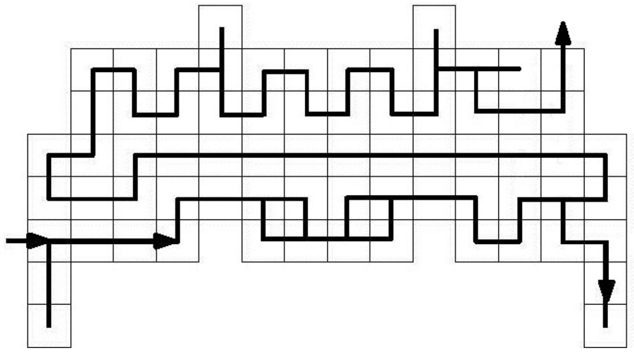


Figure 3.6: General shape of a macro-tile, for a tile types $t \in T$ and $(I, O) \in \mathrm{IO}(t)$, with $I = \{S\}$ and $s_\Sigma(t_N, t_N) < 2$.


(a) An east growing macro-tile in class 3.

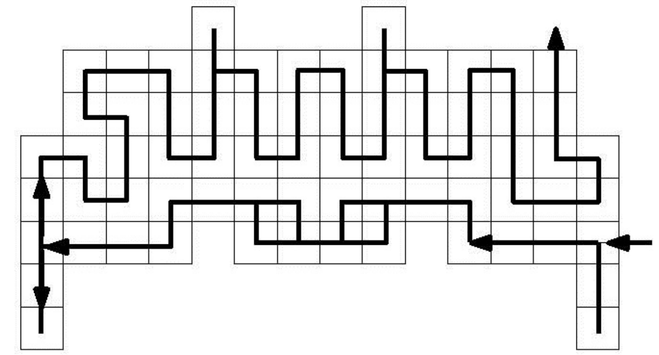

(b) A west growing macro-tile in class 3.

Figure 3.7: The macro-tiles corresponding to the copies of tiles in class 3.

Final Steps of the Proof In this part, we complete the definition of function f (see Lemma 13) by using the function Macro and we show that it satisfies the conditions of Lemma 13. Since Γ is deterministic, for a tile type $t \in T$ and an input/output side combination $(I, O) \in IO(t)$, the mapping tile in $\text{Macro}(t, (I, O))$ is unique to t and (I, O). Therefore Macro is bijective, i.e. its inverse function is well defined. Recall that,

$$T' = \bigcup_{t \in T, (I,O) \in \text{IO}(t)} \text{Macro}(t, (I, O)).$$

For each tile type $t \in T$, we created $|\,\text{IO}(t)| \leq 2$ macro-tiles and each macro-tile includes a unique tile type, the mapping tile described before, while all the other tile types in $\text{Macro}(t, (I, O))$, where $(I, O) \in \text{IO}(t)$, depend only either on the glues on sides in I or on the glues on sides in O and they might exist in other macro-tiles. Hence we have $|T'| = O(\min\{|G|^2, |T|\} \log(|T|) + 2|T|) = O(\min\{|G|^2, |T|\} \log(|T|))$. Now, function $f : \bigcup_{t \in T}\{\bigcup_{(I,O) \in \text{IO}(t)} \text{Macro}(t, (I, O))\} \mapsto T$ is defined such that for each $t \in T$,

$$f(\{\bigcup_{(I,O) \in \text{IO}(t)} \text{Macro}(t, (I, O))\}) = t.$$

In other words, for each $t \in T$ the function f maps the set of tile types in T' that assemble to $|\,\text{IO}(t)|$ macro-tiles of t, to t. For a set of tile types $A \subseteq T'$ let $\text{Map}(A)$ be the set of all mapping tiles in the set A. Then, for each $t \in T$ and $(I, O) \in \text{IO}(t)$, since the mapping tile of $\text{Macro}(t, (I, O))$ is unique to t and (I, O), we have $\text{Map}(\text{Macro}(t, (I, O))) \notin \bigcup_{t' \in T \setminus \{t\}, (I', O') \in \text{IO}(t')} \text{Macro}(t', (I', O'))$. Therefore for each $t \in T$,

$$\text{Map}(\bigcup_{(I,O) \in (\text{IO}(t))} \text{Macro}(t, (I, O))) \notin \bigcup_{t' \in T \setminus \{t\}, (I', O') \in \text{IO}(t')} \text{Macro}(t', (I', O')),$$

in other words, the set of mapping tiles of the $|\,\text{IO}(t)|$ macro-tiles of tile $t \in T$ is unique to t and this implies that f is bijective.

It remains to show that for each copy of $t \in T$ located at position (i, j) in *terminal*, there exists a set $T^* \subseteq T'$ in the domain of f with a unique tile type $t' \in T^*$ such that $\Gamma'(k, l) = t'$, for some $(4k + 6)i \leq k < (4k + 6)(i + 1)$ and $5i \leq l < 5(i + 1)$ and $f(T^*) = t$. It is tedious but not difficult to show by induction on the number of tiles in the *terminal* that if $\Gamma(i, j) = t$ then the mapping tile of the macro-tile $\text{Macro}(t, \text{Input}(t, (i, j)), \text{Output}(t, (i, j))$, is placed in the square region $(4k + 6)i \leq k < (4k + 6)(i + 1)$ and $5i \leq l < 5(i + 1)$. Since $\text{Map}(\text{Macro}(t, (I, O))) \notin \bigcup_{t' \in T} f^{-1}(t')$, for $\Gamma(i, j) = t$ we can define t' to be the $\text{Map}(\text{Macro}(t, \text{Input}(t, (i, j)), \text{Output}(t, (i, j))))$, and this completes the proof. $\square$

Note that the number of steps for assembling each macro-tile is $O(\log(|T|))$ and therefore the number of steps to assemble the terminal assembly of Γ' is linear in the number of tiles present in *terminal* with a factor $\log(|T|)$. Also, note that since for a given Turing machine M with alphabet A and input $w \in A^*$ the Zig-Zag system presented for Theorem 9 in [11] simulates each step of M in $O(|p|)$ steps, where p is the size of the tape of M at that step, therefore Γ' simulate each step of M in $O(p \log |T|)$. Now, we can prove Theorem 12.

Proof of Theorem 12. Let $\Upsilon = \Gamma'$, where Γ' is the tile system which has been constructed in the proof of Lemma 13. Let $R = \text{Map}(T')$, then for each $t' \in R$ there exists a macro-tile m where $t' = \text{Map}(m)$ and each macro-tile m. Define function SIM : $R \mapsto T$ as $\text{SIM}(t') = t$ where t is such that for some $(I, O) \in \text{IO}(t)$, $t' = \text{Map}(\text{Macro}(t, (I, O)))$. Function SIM is well defined since t' is a mapping tile. We claim SIM satisfies the conditions in Theorem 12. For each $t = \Gamma(i, j) \neq \textit{empty}$, the tile $\text{Map}(\text{Macro}(t, \text{Input}(t, (i, j)), \text{Output}(t, (i, j))))$ is unique to t and the combination $\text{Input}(t, (i, j))$ and $\text{Output}(t, (i, j))$, and is placed in the region $(4k + 6)i \leq k < (4k + 6)(i + 1)$ and $5i \leq l < 5(i + 1)$, where $k = \lceil \log(|NS|) \rceil$ and NS is defined as in proof of Lemma 13. Thus Υ simulates Γ with horizontal scale factor $4(\log(|NS|)) + 6$ and vertical scale factor 5. It is trivial that $\log(|NS|) = O(\log |T|)$. This completes the proof for Theorem 12. □

As we mentioned in Theorem 9, Zig-Zag tile assembly model at temperature 2 is capable of simulating an arbitrary Turing machine and hence is Turing Universal. Now, since in Theorem 12 we showed that Step-wise tile assembly model at temperature 1 can simulate any Zig-Zag tile assembly model at temperature 2, we can deduce the following theorem.

Theorem 18. *Step-wise tile assembly model is Turing Universal.*

3.2 Turing Universality of Staged Tile Assembly Model at Temperature 1

In this section, we prove the Turing Universality for Staged tile assembly model.

Theorem 19. *[7] For any deterministic temperature 2 Zig-Zag tile assembly system $\Gamma = \langle \Sigma, T, s, s_\Sigma, 2\rangle$ there exists a Staged tile assembly system Ω at temperature 1 that simulates Γ at vertical scale* 5*, horizontal scale* $O(\log(|T|))$ *and with tile complexity* $O(|T|\log(|T|))$.

Proof. We present a system similar to Υ in the proof for Theorem 12. Trivially, Υ can simulate Γ by using two bins, one for T'_1 and the other for T'_2, in $|\operatorname{Struc}(\operatorname{Term}(\Gamma))|\log(|T|)$ stages. In what follows we show that with using more bins in Staged assembly model, the number of stages can be reduced to 2. In the first stage, all the macro-tiles are assembled in $|T|$ bins. In the second stage, the assembled macro-tiles are mixed in an additional bin and they automatically attach together and produce the terminal assembly which is associated with $Term(\Gamma)$. In this proof we use the functions we defined in the proofs of Lemma 13 and Theorem 12.

The assembly process of macro-tiles in this system is a bit different than that used for Step-wise model, although the tiles that appear in the macro-tiles are exactly the same. For each $t \in T$, we use a distinct bin B_t at stage 1. All copies of the $|\operatorname{IO}(t)|$ macro-tiles that simulate copies of t in Ω, assemble in bin B_t. For a tile $t \in T$ and input/output side combination $(I, O) \in \operatorname{IO}(t)$, let $\operatorname{Macro}'(t, (I, O)$ be the set of tile types that assemble the macro-tile corresponding to the tile t and input/output side combination (I, O). If $\operatorname{Label}(S, t, (I, O)) = g \neq g^*$, for each b^i_g, the tile set $\operatorname{Macro}'(t, (I, O)$ includes the tile types to assemble only the correct digit b^i_g. This is different from the approach we used in $\operatorname{Macro}(t, (I, O))$ for Step-wise model. The set $\operatorname{Macro}'(t, (I, O))$ includes all other tile types in $\operatorname{Macro}(t, (I, O))$ (tile types for assembling north, east and west boundaries as well as the filler and mapping tiles).

The way in which the tile types in $\operatorname{Macro}'(t, (I, O))$ are defined guarantees that each glue is used on at most two tile types. Hence the assembly process in bin B_t proceeds in a linear way and the terminal assemblies will always be the macro-tiles for t. Algorithm 11 produces the tile types for assembling the south boundaries of the macro-tiles in B_t. We denote the tile set in bin B_t by T_{B_t}.

At stage 2, we use one addition bin, B^*, to mix the terminal assemblies from all bins

Algorithm 11 The algorithm generates the tile types in $\mathrm{Macro}'(t, I, O)$ for assembling the south boundary where $S \in I$.

```
Input: $k = \log(|NS|)$, $t \in T$ and $(I, O) \in \mathrm{IO}(t)$ such that $S \in I$
Output: Corresponding tile types in $T_{B_t}$
if $W \in I$ then
  for $i = 0$ to $k - 1$ do
    if $b_i = 0$ then
      Add the following tile types to $T_{B_t}$:
      $[\,(t_W, b_g^0 \dots b_g^{i-1}, A)\,,\ (t_W, b_g^0 \dots b_g^{i-1}, B)\,,\ \_\,,\ (t_W, b_g^0 \dots b_g^{i-1}, )\,]$
      $[\,(t_W, b_g^0 \dots b_g^{i-1}, D)\,,\ \_\,,\ \_\,,\ (t_W, b_g^0 \dots b_g^{i-1}, B)\,]$
      $[\,\_\,,\ (t_W, b_g^0 \dots b_g^{i-1}, E)\,,\ \_\,,\ \_\,]$
      $[\,\_\,,\ (t_W, b_g^0 \dots b_g^{i-1}, G)\,,\ (t_W, b_g^0 \dots b_g^{i-1}, D)\,,\ (t_W, b_g^0 \dots b_g^{i-1}, E)\,]$
      $[\,\_\,,\ (t_W, b_g^0 \dots b_g^{i-1}, 0I)\,,\ \_\,,\ (t_W, b_g^0 \dots b_g^{i-1}, G)\,]$
      $[\,\_\,,\ \_\,,\ (t_W, b_g^0 \dots b_g^{i-1}, 0F)\,,\ (t_W, b_g^0 \dots b_g^{i-1}, 0I)\,]$
      $[\,(t_E, b_g^0 \dots b_g^{i-1}, 0F)\,,\ (t_W, b_g^0 \dots b_g^{i-1}, 0)\,,\ \_\,,\ \_\,]$
    end if
    if $b_i = 1$ then
      Add the following tile types to $T_{B_t}$:
      $[\,(t_W, b_g^0 \dots b_g^{i-1}, A)\,,\ (t_W, b_g^0 \dots b_g^{i-1}, B)\,,\ \_\,,\ (t_W, b_g^0 \dots b_g^{i-1})\,]$
      $[\,\_\,,\ (t_W, b_g^0 \dots b_g^{i-1}, G)\,,\ (t_W, b_g^0 \dots b_g^{i-1}, A)\,,\ \_\,]$
      $[\,\_\,,\ (t_W, b_g^0 \dots b_g^{i-1}, 1J)\,,\ \_\,,\ (t_W, b_g^0 \dots b_g^{i-1}, G)\,]$
      $[\,\_\,,\ (t_W, b_g^0 \dots b_g^{i-1}, 1E)\,,\ (t_W, b_g^0 \dots b_g^{i-1}, 1D)\,,\ (t_W, b_g^0 \dots b_g^{i-1}, 1J)\,]$
      $[\,(t_W, b_g^0 \dots b_g^{i-1}, 1D)\,,\ (t_W, b_g^0 \dots b_g^{i-1}, 1B)\,,\ \_\,,\ \_\,]$
      $[\,\_\,,\ \_\,,\ (t_W, b_g^0 \dots b_g^{i-1}, 1I)\,,\ (t_W, b_g^0 \dots b_g^{i-1}, 1E)\,]$
      $[\,(t_W, b_g^0 \dots b_g^{i-1}, 1I)\,,\ (t_W, b_g^0 \dots b_g^{i-1}, 1)\,,\ \_\,,\ (t_W, b_g^0 \dots b_g^{i-1}, 1B)\,]$
    end if
  end for
end if
if $E \in I$ then
  Generate the same tile types as for the case $W \in I$ of this algorithm, except for the
  generated tile types flip the east and west glues and exchange $t_W$ with $t_E$ in glue labels.
end if
```

described before. In this stage, the vertical teeth on the south boundaries of macro-tiles become useful. The vertical teeth in this system, guarantee that the macro-tiles only share the right boundaries and corresponding digits of the north or south glues abut. Without these vertical horizontal slips would be possible at stage 2. Figure 3.8 illustrates a possible slip where the beginning portion of sequence corresponding to a glue in NS is the same as the end portion of binary sequence for another glue in NS.

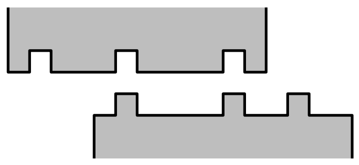


Figure 3.8: The possible slips for two macro-tiles one with south side depicting (110) the other with north side (101). The two segments of (10) can be matched together without the vertical teeth of length two on the south of macro-tiles.

We can define SIM to be a bijection between the set of all mapping tiles corresponding to t, to tile $t \in T$. With exact same arguments as we did before, one can see SIM is satisfying the conditions in Theorem 19.

$\square$

As we saw before in Theorem 9, Zig-Zag tile assembly model is capable of simulating an arbitrary Turing machine. Since Staged tile assembly model can simulate any Zig-Zag tile assembly system over an arbitrary input, we can deduce the following,

Theorem 20. *Staged tile assembly model is Turing Universal.*

Chapter 4

Triangular and Hexagonal Self-Assembly Models

In this chapter, we study the computational power of Self-Assembly systems with different tile shapes at temperature 1. In the models we studied up to this point, the basic component blocks were square tiles. In this chapter we investigate, instead of square tiles, triangular tiles and hexagonal tiles. Namely we discuss the self-assembly by equilateral-triangular and hexagonal tile systems.

4.1 Computational aspects of Triangular Tile Self-Assembly systems at Temperature 1

The Triangular tile self-assembly model is defined in a similar way to aTAM in Section 1.1. The difference is, the underlying grid is triangular. Each tile type under this model is an equilateral triangle with three sides, that is either in an upward (base facing up) or downward (base facing down) orientation, as illustrated in Figure 4.1. More formally, an *equilateral triangular* tile is represented as a quadruple (a, b, c, k), where $a, b, c \in \Sigma$ are the glues on the sides of the tile in the counter-clockwise order starting from the horizontal side and $k \in \{u, d\}$ presents the upward (u) or downward orientation (d) (see Figure 4.1). Thus, for the tile set of a triangular system we have $T_{\triangle} \subseteq \Sigma \times \Sigma \times \Sigma \times \{u, d\}$. Through this section, we will call the equilateral triangular tiles, simply triangular tiles. For a tile type $t \in T_{\triangle}$, the notations $\gamma_1(t)$, $\gamma_2(t)$ and $\gamma_3(t)$ denote the glues on the sides of t in a clockwise order starting with $\gamma_1(t)$, where $\gamma_1(t)$ is the glue on the base of the corresponding triangle.

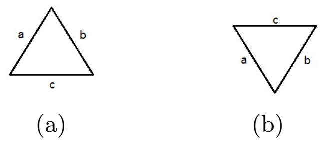
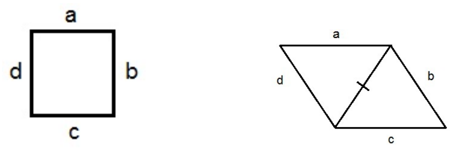


Figure 4.1: Triangular tile types can have upward or downward orientation based on the position of the horizontal side. (a) A triangular type (c, b, a, d) with downward orientation. (b) A triangular type (c, a, b, u) with upward orientation.

Theorem 21. *For any temperature 1 tile assembly system* $\Gamma = \langle \Sigma, T, s, s_\Sigma, 1\rangle$ *under aTAM there exists a Triangular tile assembly system* $\Delta = \langle \Sigma', T_\triangle, s', s'_{\Sigma'}, 1\rangle$ *at temperature 1 that simulates each square tile type of* Γ *by two triangular tile types in* Δ*.*

Proof. We use the idea of macro-tiles in Δ simulating the behavior of tiles in Γ. For each tile $t \in T$ we add two tiles t_1 and t_2 to $T_\triangle$. Since both systems are under temperature 1 the process is much simpler. Figure 4.2 illustrates the tile types in $T_\triangle$ that attach together to assemble the supertile mapped to t. Label the two tile types mapped to t by t_1 and t_2 in arbitrary order. Each tile type t_1 or t_2 is unique to t (the dash in the Figure 4.2 is a glue unique to t), thus there's a one to one mapping between the pair of tile types t_1 and t_2 in $t_\triangle$ and $t \in T$. Thus we define a bijective function SIM : $\bigcup_{t\in T}\{t_1 : \text{Macro}(t) = \{t_1, t_2\}\} \mapsto T$ as $\text{SIM}(t_1) = t$. Note that $|T_\triangle| = 2|T|$.



(a) A tile $t \in T$.

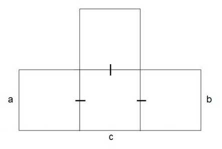
(b) The tiles in $T_\triangle$ simulating t.

Figure 4.2: The supertile simulating tile $t \in T$ in Δ. The dashes represent distinct glues that are unique to t.

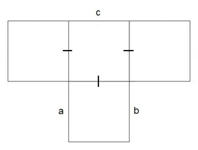
□

Theorem 22. *For any temperature 1 Triangular tile assembly system* $\Delta = \langle \Sigma', T_\triangle, s', s'_{\Sigma'}, 1\rangle$ *there exists a tile assembly system* $\Gamma = \langle \Sigma, T, s, s_\Sigma, 1\rangle$ *under aTAM at temperature 1 that simulates each triangular tile type in* Δ *by 4 square tile types in* Γ*.*

Proof. For each tile types $t \in T_\triangle$, we design a macro-tile to simulate t in Γ. The unique tile types in T which simulate a tile type $t \in T_\triangle$ are pictured in Figure 4.3. We can describe the function SIM, in a similar way as before, to map a fixed tile type from the set of 9 tile types in T that assemble to the macro-tile for $t \in T_\triangle$, to t. Note that the dashes in Figure 4.3 are unique to t and therefore all the tile types in T that simulate a tile type $t \in T_\triangle$ are unique to t. □



(a) The supertile in Γ simulating the tile (c, b, a, d) in Δ pictured in Figure 4.1(a)



(b) The supertile in Γ simulating the tile (c, a, b, u) in Δ pictured in Figure 4.1(b)

Figure 4.3: The supertile simulating a tile type $t \in T_\triangle$ in Γ. The dashes represent distinct glues that are unique to t.

Consequently, from computational point of view Triangular tile assembly model is equivalent to aTAM at temperature 1.

4.2 Computational aspects of Hexagonal Tile Self-Assembly systems at Temperature 1

The Hexagonal tile self-assembly model is similar to aTAM, described in Section 1.1. Again, here the difference is that in this model we will be working with a hexagonal grid. Each tile type under this model is a regular hexagon. Without loss of generality, we assume that all regular hexagonal tiles are oriented as illustrated in Figure 4.4. More formally, a *regular hexagonal* tile is represented as a 6-tuple (a, b, c, d, e, f), where $a, b, c, d, e, f \in \Sigma$ are the glues on the sides of the tile in the clockwise order starting from the top horizontal side. Therefore, for the tile set of a Hexagonal system we have $T_{HEX} \subseteq \Sigma^6$. Through this section, we will call the regular hexagonal tiles, simply hexagonal tiles.

Theorem 23. *For any temperature 1 tile assembly system* $\Gamma = \langle \Sigma, T, s, s_\Sigma, 1 \rangle$ *under aTAM*

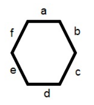


Figure 4.4: A tile type (a, b, c, d, e, f) in T_{HEX}

there exists a Hexagonal tile assembly system $\chi = \langle \Sigma', T_{HEX}, s', s'_{\Sigma'}, 1 \rangle$ at temperature 1 that simulates a square tile type in Γ with four hexagonal tile types in χ.

Proof. The supertiles in χ that simulates a tile types $t \in T$ is illustrated in Figure 4.5. We can define SIM in the same way as we did in previous section (Section 4.1). $\square$

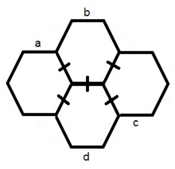


Figure 4.5: The macro-tile in χ simulating a tile type (b, c, d, a) in T

Theorem 24. *For any deterministic temperature 1 tile assembly system $\chi = \langle \Sigma', T_{HEX}, s', s'_{\Sigma'}, 1 \rangle$ under Hexagonal tile assembly model there exists a deterministic Standard tile assembly system $\Gamma = \langle \Sigma, T, s, s_{\Sigma}, 1 \rangle$ at temperature 1 that simulates χ with tile complexity $O(|T|)$.*

Proof. The supertiles in Γ that simulates a tile types $t \in T_{HEX}$ is illustrated in Figure 4.6. Figure 4.2 pictures how Γ simulates a sample super tile in χ. The function SIM can be defined in the same way as previous section.

$\square$

Thus, the Hexagonal tile assembly model has the same computational power as aTAM at temperature 1.

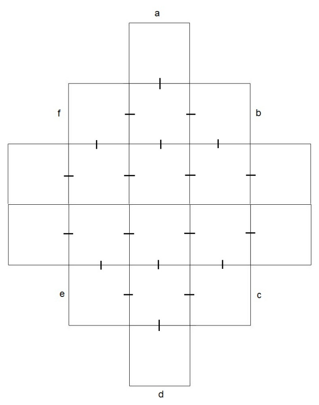
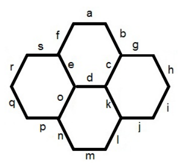
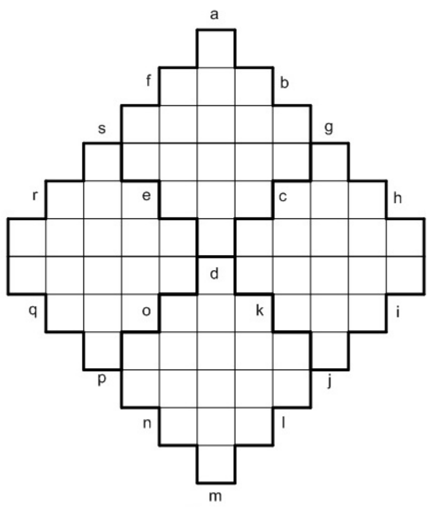


Figure 4.6: The supertile in Γ simulating a tile type (a, b, c, d, e, f) in T_{HEX}



(a) A supertile in χ consisting of 4 hexagonal tiles attached together.



(b) The super tile under Γ which simulates the super tile illustrated in part (a).

Figure 4.7: Simulating a sample supertile with 4 hexagonal tiles in χ by 4×18 square tiles in Γ.

Bibliography

[1] L. Adleman, Q. Cheng, A. Goel, and M.D. Huang. Running time and program size for self-assembled squares. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 740–748. ACM, 2001.

[2] L. Adleman, Q. Cheng, A. Goel, M.D. Huang, D. Kempe, P.M. De Espanes, and P.W.K. Rothemund. Combinatorial optimization problems in self-assembly. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 23–32. ACM, 2002.

[3] L. Adleman, J. Kari, L. Kari, D. Reishus, and P. Sosik. The undecidability of the infinite ribbon problem: Implications for computing by self-assembly. *SIAM Journal on Computing*, 38(6):2356–2381, 2009.

[4] L.M. Adleman et al. Molecular computation of solutions to combinatorial problems. *SCIENCE-NEW YORK THEN WASHINGTON-*, pages 1021–1021, 1994.

[5] G. Aggarwal, Q. Cheng, M.H. Goldwasser, M.Y. Kao, P.M. de Espanes, and R.T. Schweller. Complexities for generalized models of self-assembly. *SIAM Journal on Computing*, 34(6):1493–1515, 2005.

[6] David Barker-Plummer. Turing machines. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Winter 2012 edition, 2012.

[7] B. Behsaz, J. Maňuch, and L. Stacho. Turing universality of step-wise and stage assembly at temperature 1. *DNA Computing and Molecular Programming*, pages 1–11, 2012.

[8] T.L. Breen, J. Tien, RJ Scott, T. Hadzic, G.M. Whitesides, et al. Design and self-assembly of open, regular, 3d mesostructures. *Science*, 284(5416):948–951, 1999.

[9] H.L. Chen and D. Doty. Parallelism and time in hierarchical self-assembly. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1163–1182. SIAM, 2012.

[10] H.L. Chen, R. Schulman, A. Goel, and E. Winfree. Reducing facet nucleation during algorithmic self-assembly. *Nano Letters*, 7(9):2913–2919, 2007.

[11] Matthew Cook, Yunhui Fu, and Robert Schweller. Temperature 1 self-assembly: deterministic assembly in 3d and probabilistic assembly in 2d. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pages 570–589. SIAM, 2011.

[12] E.D. Demaine, M.L. Demaine, S.P. Fekete, M. Ishaque, E. Rafalin, R.T. Schweller, and D.L. Souvaine. Staged self-assembly: nanomanufacture of arbitrary shapes with o (1) glues. *Natural Computing*, 7(3):347–370, 2008.

[13] Baoquan Ding, Ruojie Sha, and Nadrian C Seeman. Pseudohexagonal 2d dna crystals from double crossover cohesion. *Journal of the American Chemical Society*, 126(33):10230–10231, 2004.

[14] D. Doty, M.J. Patitz, D. Reishus, R.T. Schweller, and S.M. Summers. Strong fault-tolerance for self-assembly with fuzzy temperature. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 417–426. IEEE, 2010.

[15] D. Doty, M.J. Patitz, and S.M. Summers. Limitations of self-assembly at temperature 1. *Theoretical Computer Science*, 412(1):145–158, 2011.

[16] David Doty. Theory of algorithmic self-assembly. *Communications of the ACM*, 55(12):78–88, 2012.

[17] S.M. Douglas, H. Dietz, T. Liedl, B. Högberg, F. Graf, and W.M. Shih. Self-assembly of dna into nanoscale three-dimensional shapes. *Nature*, 459(7245):414–418, 2009.

[18] Yunhui Fu and Robert T. Schweller. Temperature 1 self-assembly: Deterministic assembly in 3d and probabilistic assembly in 2d. *CoRR*, abs/0912.0027, 2009.

[19] K. Fujibayashi, R. Hariadi, S.H. Park, E. Winfree, and S. Murata. Toward reliable algorithmic self-assembly of dna tiles: A fixed-width cellular automaton pattern. *Nano Letters*, 8(7):1791–1797, 2007.

[20] D. Han, S. Pal, J. Nangreave, Z. Deng, Y. Liu, and H. Yan. Dna origami with complex curvatures in three-dimensional space. *Science*, 332(6027):342–346, 2011.

[21] Yu He, Yi Chen, Haipeng Liu, Alexander E Ribbe, and Chengde Mao. Self-assembly of hexagonal dna two-dimensional (2d) arrays. *Journal of the American Chemical Society*, 127(35):12202–12203, 2005.

[22] Rolf Herken. *The universal Turing machine: a half-century survey*. Springer Verlag Wien, 1995.

[23] M.Y. Kao and R. Schweller. Reducing tile complexity for self-assembly through temperature programming. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 571–580. ACM, 2006.

[24] L. Kari, S. Seki, and Z. Xu. Triangular and hexagonal tile self-assembly systems. *Computation, Physics and Beyond*, pages 357–375, 2012.

[25] Lila Kari, Shinnosuke Seki, and Zhi Xu. Triangular and hexagonal tile self-assembly systems. In *Proceedings of the 2012 international conference on Theoretical Computer Science: computation, physics and beyond*, WTCS'12, pages 357–375, Berlin, Heidelberg, 2012. Springer-Verlag.

[26] J. Lathrop, J. Lutz, M. Patitz, and S. Summers. Computability and complexity in self-assembly. *Logic and Theory of Algorithms*, pages 349–358, 2008.

[27] Dage Liu, Mingsheng Wang, Zhaoxiang Deng, Richard Walulu, and Chengde Mao. Tensegrity: construction of rigid dna triangles with flexible four-arm dna junctions. *Journal of the American Chemical Society*, 126(8):2324–2325, 2004.

[28] K. Lund, A.J. Manzo, N. Dabby, N. Michelotti, A. Johnson-Buck, J. Nangreave, S. Taylor, R. Pei, M.N. Stojanovic, N.G. Walter, et al. Molecular robots guided by prescriptive landscapes. *Nature*, 465(7295):206–210, 2010.

[29] Ján Manuch, Ladislav Stacho, and Christine Stoll. Step-assembly with a constant number of tile types. In *ISAAC*, pages 954–963, 2009.

[30] Matthew J. Patitz, Robert T. Schweller, and Scott M. Summers. Exact shapes and turing universality at temperature 1 with a single negative glue. In *Proceedings of the 17th international conference on DNA computing and molecular programming*, DNA'11, pages 175–189, Berlin, Heidelberg, 2011. Springer-Verlag.

[31] J.A. Pelesko. *Self assembly: the science of things that put themselves together*. Chapman & Hall/CRC, 2007.

[32] L. Qian and E. Winfree. Scaling up digital circuit computation with dna strand displacement cascades. *Science*, 332(6034):1196–1201, 2011.

[33] L. Qian, E. Winfree, and J. Bruck. Neural network computation with dna strand displacement cascades. *Nature*, 475(7356):368–372, 2011.

[34] J.H. Reif. Local parallel biomolecular computation. *DNA-Based Computers*, 3:217–254, 1999.

[35] P.W.K. Rothemund and E. Winfree. The program-size complexity of self-assembled squares. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 459–468. ACM, 2000.

[36] N.C. Seeman. Nucleic acid junctions and lattices. *Journal of theoretical biology*, 99(2):237–247, 1982.

[37] N.C. Seeman. An overview of structural dna nanotechnology. *Molecular biotechnology*, 37(3):246–257, 2007.

[38] M. Sipser. *Introduction to the Theory of Computation*, volume 27. Thomson Course Technology Boston, MA, 2006.

[39] D. Soloveichik and E. Winfree. Complexity of compact proofreading for self-assembled patterns. *DNA Computing*, pages 305–324, 2006.

[40] D. Soloveichik and E. Winfree. Complexity of self-assembled shapes. *SIAM Journal on Computing*, 36(6):1544–1569, 2007.

[41] C. Stoll. *Bounds on the tile complexity of shapes in self-assembly systems.* PhD thesis, Dept. of Mathematics-Simon Fraser University, 2009.

[42] H. Wang. *Proving theorems by pattern recognition-II.* American Telephone and Telegraph Company, 1961.

[43] E. Winfree. *Algorithmic self-assembly of DNA.* PhD thesis, California Institute of Technology, 1998.

[44] E. Winfree, F. Liu, L.A. Wenzler, N.C. Seeman, et al. Design and self-assembly of two-dimensional dna crystals. *Nature*, 394(6693):539–544, 1998.

[45] Erik Winfree and Renat Bekbolatov. Proofreading tile sets: Error correction for algorithmic self-assembly. In *DNA computing*, pages 126–144. Springer, 2004.

[46] B. Yurke, A.J. Turberfield, A.P. Mills, F.C. Simmel, and J.L. Neumann. A dna-fuelled molecular machine made of dna. *Nature*, 406(6796):605–608, 2000.