Universal Simulation of Directed Systems in the abstract Tile Assembly Model Requires Undirectedness

Jacob Hendricks
Dept of Computer Science and Information Systems
University of Wisconsin - River Falls
River Falls, WI, USA
jacob.hendricks@uwrf.edu

Matthew J. Patitz, Trent A. Rogers
Dept of Computer Science and Computer Engineering
University of Arkansas
Fayetteville, AR, USA
{patitz,tar003}@uark.edu

***Abstract*—As a mathematical model of tile-based self-assembling systems, Winfree's abstract Tile Assembly Model (aTAM) has proven to be a remarkable platform for studying and understanding the behaviors and powers of self-assembling systems. Furthermore, as it is capable of Turing universal computation, the aTAM allows algorithmic self-assembly, in which the components can be designed so that the rules governing their behaviors force them to inherently execute prescribed algorithms as they combine. This power has yielded a wide variety of theoretical results in the aTAM utilizing algorithmic self-assembly to design systems capable of performing complex computations and forming extremely intricate structures. Adding to the completeness of the model, in FOCS 2012 the aTAM was shown to also be *intrinsically universal*, which means that there exists one single tile set such that for any arbitrary input aTAM system, that tile set can be configured into a "seed" structure which will then cause self-assembly using that tile set to simulate the input system, capturing its full dynamics modulo only a scale factor. However, the "universal simulator" of that result makes use of nondeterminism in terms of the tiles placed in several key locations when different assembly sequences are followed. This nondeterminism remains even when the simulator is simulating a system which is *directed*, meaning that it has exactly one unique terminal assembly and for any given location, no matter which assembly sequence is followed, the same tile type is always placed there. The question which then arose was whether or not that nondeterminism is fundamentally required, and if any universal simulator must in fact utilize more nondeterminism than directed systems when simulating them.**

In this paper, we answer that question in the affirmative: the class of directed systems in the aTAM is not intrinsically universal, meaning there is no universal simulator for directed systems which itself is always directed. This result provides a powerful insight into the role of nondeterminism in self-assembly, which is itself a fundamentally nondeterministic process occurring via unguided local interactions. Furthermore, to achieve this result we leverage powerful results of computational complexity hierarchies, including tight bounds on both best and worst-case complexities of decidable languages, to tailor design systems with precisely controllable space resources available to computations embedded within them. We also develop novel techniques for designing systems containing subsystems with disjoint, mutually exclusive computational powers. The main result will be important in the development of future simulation systems, and the supporting design techniques and lemmas will provide powerful tools for the development of future aTAM systems as well as proofs of their computational abilities.

I. Introduction

Self-assembly is the process by which relatively simple components begin in a disorganized state and, without external guidance but only by following local rules of interaction, autonomously combine to form more complex structures. Self-assembling systems are ubiquitous in nature, and self-assembly processes govern the formation of everything from ice crystals to cellular membranes, and despite the seemingly random nature of these systems, they serve as a ratchet for the generation of complexity on scales from the nano [1], [2] to the macro [3]. The random motions of components are leveraged to allow binding opportunities to growing structures, and if the dynamics of interactions fall into ranges which are restrictive enough, without being too restrictive, ordered assemblies can form. Clearly, nondeterminism plays key roles in such systems, and our main result helps to elucidate one of them.

The abstract Tile Assembly Model (aTAM) is a mathematical abstraction of self-assembling systems based on square "tile" components which have "glues" on their sides that allow them to bind together when glues on abutting edges of tiles have matching types. Despite being a very simplified model which uses geometrically basic building blocks, the aTAM is computationally universal [4] and a powerful model allowing for very efficient algorithmic self-assembly of shapes [5], [6]. Another noteworthy aspect of the model is that it is intrinsically universal (IU) [7], meaning that there exists a single tile set, U, such that given any arbitrary aTAM system $\mathcal{T}$, U can be given an initial configuration which will cause it to faithfully simulate the full dynamics of $\mathcal{T}$ modulo a constant scale factor (dependent on $\mathcal{T}$). Since the result of [7], several other results related to IU have been used to examine and classify the relative powers of a variety of models of self-assembly and classes of systems within them [8]–[16], thus developing a complexity hierarchy which can be used to categorize models and systems within them.

In this paper, we investigate the problem of characterizing the role of nondeterminism within the aTAM, which has previously been explored in a variety of different aspects [17]–[19]. At its core, the aTAM is an asynchronous and nondeterministic model in which tile attachments to a growing assembly, while constrained by the requirement that sufficient matching glues must bind, are random with respect to the sequence of locations and sometimes the particular types of tiles which bind. The amount of nondeterminism of different aTAM systems can vary wildly, with some systems having uncountably infinite sets of producible, or even terminal (i.e. those which cannot grow any further), assemblies and/or sequences of assembly, to those having exactly one producible assembly and even some with just one possible assembly sequence. This leads to questions about whether or not, and possibly how much, nondeterminism is required to give the aTAM its full power. In this paper, we focus on this question from the perspective of the "universal aTAM simulator" of [7], which by design has several so-called "points of competition", where different assembly sequences of the simulator, as it simulates a system $\mathcal{T}$, race to grow paths to those points, with the first path to arrive causing a tile type specific to that path to be placed. The fact that there are multiple assembly sequences, each growing a different path first, causes nondeterminism in the types of tiles placed in these locations. The use of such locations is so fundamental to that universal simulator's design, allowing it to continue growth of portions of the assembly without having to rely on future paths which may or may not ever arrive, that even when it is simulating directed aTAM systems, which are those that have exactly one terminal assembly and only one possible tile type in any location regardless of the assembly sequence, the simulator itself must be undirected. It has remained unknown whether or not such nondeterminism is fundamentally required by a universal simulator, and in Theorem IV.1 we prove that it is. That is, we prove that the class containing all directed aTAM systems is not IU, meaning that there exists no tile set U such that, given an arbitrary directed aTAM system, U can be configured to create an aTAM system which simulates it while itself being directed. Stated another way, it means that any universal simulator for the aTAM must be more nondeterministic than some of the systems which it simulates.

While our main result presents key insights into the properties required of aTAM and other tile-based simulators, and shows how nondeterminism with respect to the selection of assembly sequences can force nondeterminism with respect to assemblies produced by any universal simulator, other key contributions of this paper include the development of several new system design techniques and tools useful in proving properties about the computational resources available to be harnessed by embedded algorithms, which themselves provide additional insights into the computations possible using static combinations of matter filling non-reusable space. More specifically, we make use of computational complexity results which combine extremely tight worst-case and best-case space complexity bounds for decidable languages [20], as well as novel techniques for controlling the "input bandwidth" and geometries of carefully designed subassemblies which perform complex computations that are effectively hidden from each other. These designs are likely to be useful in further tile-based self-assembly results, especially impossibility results. Furthermore, we develop several important and potentially very useful tools which can be used to characterize properties of tile assembly systems which are simulating others, e.g. Lemma III.2 which proves that the space complexity of computations which can be performed by a system simulating a type of system known as a zig-zag system is asymptotically no greater than that of the computations which can be performed by the original system, despite the scale factor allowed the simulator.

Section II provides a set of preliminary definitions used throughout the paper, including those of the aTAM, simulation, intrinsic universality, and zig-zag systems. The following section provides a few more definitions specific to some of our technical lemmas, and then provides the statement of our main technical lemma. Section IV contains the formal statement of our main result, and next are two sections dedicated to a high-level overview of its proof. Due to space constraints full technical details can be found in [21].

II. Preliminaries

In this section we provide an informal definition of the aTAM and then define what it means for one tile assembly system to simulate another, and the notion of intrinsic universality. We also provide the definition of zig-zag systems.

A. Informal description of the abstract Tile Assembly Model

This section gives a brief informal sketch of the abstract Tile Assembly Model (aTAM). See [21] for a formal definition of the aTAM.

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

Each assembly induces a *binding graph*, a grid graph whose vertices are tiles, with an edge between two tiles if they interact. The assembly is *τ-stable* if every cut of its binding graph has strength at least τ, where the strength of a cut is the sum of all of the individual glue strengths in the cut.

A *tile assembly system* (TAS) is a triple $\mathcal{T} = (T, \sigma, \tau)$, where T is a finite set of tile types, $\sigma : \mathbb{Z}^2 \dashrightarrow T$ is a finite, τ-stable *seed assembly*, and τ is the *temperature*. An assembly α is *producible* if either $\alpha = \sigma$ or if β is a producible assembly and α can be obtained from β by the stable binding of a single tile. In this case we write $\beta \to_1^{\mathcal{T}} \alpha$ (to mean α is producible from β by the attachment of one tile), and we write $\beta \to^{\mathcal{T}} \alpha$ if $\beta \to_1^{\mathcal{T}*} \alpha$ (to mean α is producible from β by the attachment of zero or more tiles). When $\mathcal{T}$ is clear from context, we may write $\to_1$ and $\to$ instead. We let $\mathcal{A}[\mathcal{T}]$ denote the set of producible assemblies of $\mathcal{T}$. An assembly is *terminal* if no tile can be τ-stably attached to it. We let $\mathcal{A}_\Box[\mathcal{T}] \subseteq \mathcal{A}[\mathcal{T}]$ denote the set of producible, terminal assemblies of $\mathcal{T}$. A TAS $\mathcal{T}$ is *directed* if $|\mathcal{A}_\Box[\mathcal{T}]| = 1$. Hence, although a directed system may be nondeterministic in terms of the order of tile placements, it is deterministic in the sense that exactly one terminal assembly is producible (this is analogous to the notion of *confluence* in rewriting systems).

B. Simulation

To state our main results, we must formally define what it means for one TAS to "simulate" another. Our definitions come from [13]. Intuitively, simulation of a system $\mathcal{T}$ by a system $\mathcal{S}$ requires that there is some scale factor $m \in \mathbb{Z}^+$ such that $m \times m$ squares of tiles in $\mathcal{S}$ represent individual tiles in $\mathcal{T}$, and there is a "representation function" capable of inspecting assemblies in $\mathcal{S}$ and mapping them to assemblies in $\mathcal{T}$.

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

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

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

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

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

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

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

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

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

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

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

C. Intrinsic Universality

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

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

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

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

Definition II.6. *We say that U is* intrinsically universal *for $\mathfrak{C}$ if it is intrinsically universal for $\mathfrak{C}$ at some temperature $\tau' \in Z^+$.*

Definition II.7. *We say that $\mathfrak{C}$ is intrinsically universal if there exists some U that is intrinsically universal for $\mathfrak{C}$ and for every $\mathcal{T} \in \mathfrak{C}$ and $\mathcal{U}_\mathcal{T}$ which simulates it, $\mathcal{U}_\mathcal{T} \in \mathfrak{C}$.*

D. Zig-zag assembly systems

In [22], a system $\mathcal{T} = (T, \sigma, \tau)$ is called a zig-zag tile assembly system provided that (1) $\mathcal{T}$ is directed, (2) there is a single sequence $\vec{\alpha} \in \mathcal{T}$ with $\mathcal{A}_\Box[\mathcal{T}] = \{\vec{\alpha}\}$, and (3) for every $\vec{x} \in \text{dom } \alpha, (0, 1) \notin \text{IN}^{\vec{\alpha}}(\vec{x})$. We say that an assembly sequences satisfying (2) and (3) is a *zig-zag assembly sequence*. Intuitively, a zig-zag tile assembly system is a system which grows to the left or right, grows up some amount, and then continues growth again to the left or right. Again, as defined in [22], we call a tile assembly system $\mathcal{T} = (T, \sigma, \tau)$ a *compact zig-zag tile assembly system* if and only if $\mathcal{A}_\Box[\mathcal{T}] = \{\vec{\alpha}\}$ and for every $\vec{x} \in \text{dom } \alpha$ and every $\vec{u} \in U_2$, $\text{str}_{\alpha(\vec{x})}(\vec{u}) + \text{str}_{\alpha(\vec{x})}(-\vec{u}) < 2\tau$. Informally, this can be thought of as a zig-zag tile assembly system which is only able to travel upwards one tile at a time before being required to zig-zag again. The assembly sequence of a compact zig-zag system is called a *compact zig-zag assembly sequence*. Figure 1 depicts a compact zig-zag assembly sequence. As in the definition of a zig-zag system and throughout this section, we assume that each row of a zig-zag systems binds to the north of the previous row.

III. Space complexity of zig-zag systems is invariant under simulation

In this section, we give a formal definition of a language defined by a zig-zag system. We next show that such a language can be computed in space on the order of the maximal width of the zig-zag assembly grown to a finite height. While this result is fairly straightforward, we include it for the sake of completeness and because it serves as the basis of the main result of this section (Lemma III.2). We give a formal definition of a language defined by a simulation of a zig-zag system. Lemma III.2 states that even though such a language is defined in terms of a simulator of a zig-zag system, such a language can be computed in space on the order of the maximal width of the zig-zag assembly grown to a finite height in the simulated system. In other words, even though the system simulating a zig-zag system does not have to follow the dynamics of a zig-zag system as m-block supertiles assemble and as tiles assemble in fuzz regions (see Section II), the defined language can still be computed in space on the order of the maximal width of the zig-zag assembly grown to a finite height in the simulated system.

Here is some of the notation used in this section. Let $\mathcal{T} = (T, \sigma, \tau)$ be a temperature τ compact zig-zag system with a seed σ consisting of a single tile, and let α be an assembly in $\mathcal{A}[\mathcal{T}]$. Since all of the results in this section hold regardless of the location of σ, without loss of generality, throughout this section, we assume that the location of σ is $(0, 0)$.

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

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

Now, let $f : \mathbb{N} \to \mathbb{N}$ be the function $f(n) = \max\{w_j \mid w_j \text{ is the width of the } j^{th} \text{ row of } \alpha \text{ for } 0 \le j \le n\}$.

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

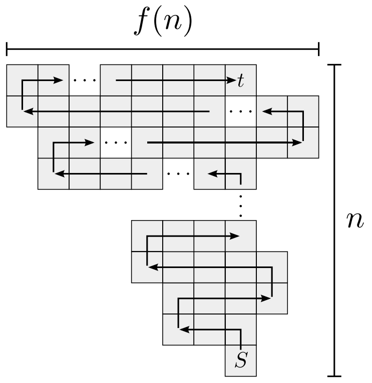

Figure 1: An assembly with a zig-zag assembly sequence. The assembly sequence is indicated with arrows. The tile labeled S makes up the seed σ, and $r(n) = 1$ if and only if the tile type of the tile labeled t is in T_1.

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

Lemma III.1. *Let $\mathcal{T} = (T, \sigma, \tau)$ be a zig-zag system with tile set T, seed assembly σ, and temperature τ. Let T_1 be* *some subset of T, let r be the characteristic function for $\mathcal{T}$ given T_1, and let L_r be the language defined by $\mathcal{T}$ given r. Finally, let $f(n)$ denote the width of the longest row of the assembly of $\mathcal{T}$ consisting of n completed rows. Then, $L_r \in \mathsf{DSPACE}(f(n))$.*

To generalize Lemma III.1, we require a definition of a language defined by a *simulation* of a zig-zag system which we describe here. Let $\mathcal{S} = (S, \sigma_S, \tau')$ be a TAS that simulates $\mathcal{T}$ with representation function R and scale factor c. Since all of the results here hold upto translation of assemblies in $\mathbb{Z}^2$, without loss of generality, throughout this section we assume that the bottom-right tile of σ_S has location $(0,0)$. The following notation is similar to the notation used for stating Lemma III.1, only it is generalized to pertain to simulations of zig-zag systems. Let α'_n denote the subassembly of α' such that for all $(i,j) \in \mathrm{dom}\ (\alpha'_n)$, $j \leq n$ and for all $(i,j) \in \partial^{\tau'} \alpha'_n$, $j \geq n+1$. For some $L \subset \mathbb{Z}^2$ such that $|L| < \infty$ and for $\vec{v} \in \mathbb{Z}^2$, let $L_{\vec{v}}$ denote $\{\vec{l} + \vec{v} \mid \vec{l} \in L\}$. Also, for $\vec{n} = (0,n)$, let $C_n \subseteq \{w \mid w : L_{\vec{n}} \to S$ is a partial function$\}$ be a subset of configurations (i.e. partial functions from $\mathbb{Z}^2$ to T) over S with domain in $L_{\vec{n}}$. Then we let $r' : \mathbb{N} \to \{0,1\}$ be the function

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

Essentially, r' is the function obtained by growing α' to the point where the next tile added must be at a location above the line $y = n$, and then considering some finite configuration (i.e. a partial functions from $\mathbb{Z}^2$ to T with finite domain) of this assembly. $r'(n) = 1$ if and only if this configuration is in C_n. Finally, let $L'_r = \{n \in \mathbb{N} \mid r'(n) = 1\}$. As with zig-zag systems, we call C_n a set of finite configurations, r' the *characteristic function for $\mathcal{S}$ given C_n*, and $L_{r'}$ the *language defined by $\mathcal{S}$ given r'*. Notice that r' is a computable function and $L_{r'}$ is a computable set.

Lemma III.2, which we refer to as the "No Cheating Lemma", states that a system which simulates a zig-zag system, even though it is allowed a scale factor, is allowed to place the individual tiles of macrotiles in varying orders, and can perhaps utilize fuzz regions for additional communication and computation, nonetheless has the same asymptotic upper bound on the space complexity of its language as the original system which it is simulating.

Lemma III.2 (No Cheating Lemma)**.** *Let $\mathcal{T} = (T, \sigma, \tau)$ be a zig-zag system and let $\mathcal{S} = (S, \sigma_S, \tau')$ be a system that simulates $\mathcal{T}$ at temperature τ' with scale factor c. Let n be in $\mathbb{N}$, and let $f(n)$ be the width of the longest row of the assembly of $\mathcal{T}$ consisting of n completed rows. Moreover, let C_{cn} be a set of finite configurations, let r' be the characteristic function for $\mathcal{S}$ given C_{cn}, and let $L_{r'}$ be the language defined by $\mathcal{S}$ given r'. Then, $L_{r'} \in \mathsf{DSPACE}(f(n))$.*

Lemma III.2, an important tool in the proof of our main theorem, implies that the constraints imposed by the dynamics of correct simulation of a zig-zag system are such that the simulating system cannot perform computations which require asymptotically more space than the original system. Intuitively, the proof, which can be found in [21], shows that the requirement to form macrotiles in $\mathcal{S}$ which map to tiles in $\mathcal{T}$ in the same order as tiles appear in $\mathcal{T}$, while only growing allowable fuzz, forces the simulator to use space which is restricted by the size of the rows in $\mathcal{T}$ multiplied by the scale factor of the simulation, which is a constant. Therefore, this lemma provides an important tool for equating the computational power of simulators and simulated zig-zag systems.

IV. The Directed aTAM is not Intrinsically Universal

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

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

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

Our proof of Theorem IV.1 will be by contradiction. Therefore, assume that such a universal tile set U, which can be used to simulate any directed system while using a directed system, exists. Given that U, we define an aTAM system $\mathcal{T} = (T, \sigma, 2)$ which is directed and forms an infinite terminal assembly, explain the growth of $\mathcal{T}$, and verify that it is directed. We provide a high-level overview of $\mathcal{T}$ in Section V. We then show why there exists no directed aTAM system $\mathcal{S} = (U, \sigma_{\mathcal{T}}, \tau')$ which simulates $\mathcal{T}$. Section VI contains a very high-level overview of that proof. Full details of $\mathcal{T}$ and of the impossibility proof can be found in [21].

V. Overview of the Directed aTAM System $\mathcal{T}$

At the highest level, $\mathcal{T}$ self-assembles an infinite structure, starting from a single seed tile placed at the origin, and growing from left to right. In well-defined intervals, as the assembly grows eastward it initiates upward growths, an infinite series of sets of three "modules" which are subassemblies able to grow almost entirely independently of each other once the main horizontal growing structure has placed the tiles which serve as the "input" for the growth of each. The aTAM is computationally universal [4], and in fact it is quite straightforward to design a tile assembly system which simulates the computation of an

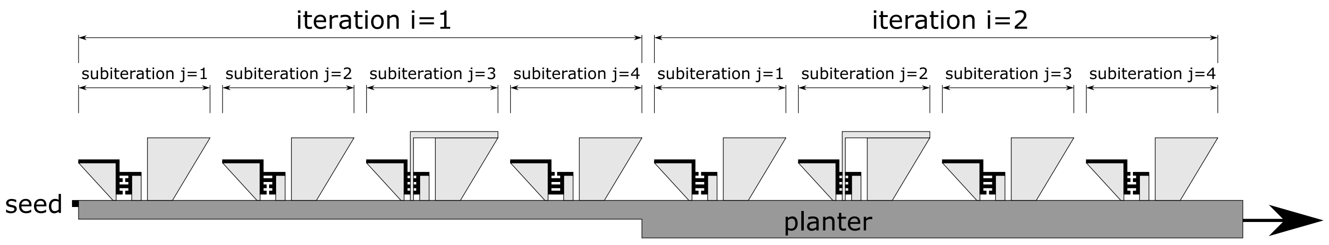

Figure 2: A high-level schematic depiction of a portion of the infinite assembly produced by a directed aTAM system $\mathcal{T}$ which cannot be simulated by any directed universal simulator.

arbitrary Turing machine M (e.g. [23], [24]) by growing rows of tiles, one above the other, where each row represents the full configuration of M at a given time step (i.e. the tape contents, read/write head location, and state) in the values of the glues encoded on their north sides, and the row immediately above it represents the full configuration of M at the next time step (by designing the tile types appropriately so that the only tiles which can attach above a given row ensure that the new northern glue above a position which just had the read/write head encodes the value that would have been output given the state of M and the cell's previous value, and depending on the direction the head would have moved, either the tile representing the cell to the left or write would have a glue encoding the new state of M and the current value of that cell). To provide a logically infinite tape, the tiles can be designed to grow rows "on demand" by extending a row by one tile each time the simulated read/write head attempts to move past the end of the currently represented row.

The three modules which grow upward are logically grouped so that there is one of each type in a set. These three modules are designed so that they simulate three computations which require asymptotically differing space resources. As each set is initiated with inputs of increasing values, and as the assembly grows infinitely to the right, those space requirements ensure that the smallest module cannot perform the computations of the larger two, and the mid-sized module cannot perform the computations of the largest. The computations carried out by each set of grouped modules as well as the geometries to which they are each constrained are carefully designed such that two of the modules are necessarily completely "ignorant" of the eventual outputs of the others. However, these two modules are designed so that after performing their computations, they grow assemblies representing bit strings corresponding to the outputs of their computations in locations across a one tile wide gap from each other, which we call the `bitAlley`. In locations where output bits of the two computations match, tiles attach between tiles for those bit positions. The third module independently computes the results of the computations of both other modules and if and only if there will be no matching bits between them, it grows an assembly which is a single tile wide path down through the `bitAlley` (thus it is guaranteed not to crash into any tiles in the `bitAlley`, regardless of the ordering of tile attachments). As the overall assembly grows further right, the inputs to the modules increase and the computations simulated by the modules require more resources and the `bitAlleys` become arbitrarily long. We are able to first show that $\mathcal{T}$ is directed, and then that no simulating system can be built using the tiles of a universal simulating tile set U and be itself directed. This is because any such directed simulator is forced by the dynamics of correct simulation, the mutual obfuscation of computations across modules, and geometric constraints, to effectively create bottlenecks which do not allow enough information to be transmitted to the growing assembly for correct growth and therefore simulation. The intuition is that the simulator has to make "guesses" about when it may need to place tiles which cooperate across a `bitAlley` (i.e. glues from the tiles on both sides of the gap are required to allow the attachment of one between them) which, due to the fact that space cannot be reused in the aTAM, doom it to failure. Furthermore, these guesses are required not by nondeterminism about which tiles can be placed in locations by $\mathcal{T}$, since after all $\mathcal{T}$ is directed, but rather due to the ordering of arrival of tiles - the particular assembly sequence which may be followed.

A. *Overview of modules of $\mathcal{T}$*

Figure 2 shows a schematic depiction of a portion of the terminal assembly of $\mathcal{T}$. We now give a very high-level description of each of the main modules, and full details can be found in [21].

Beginning from the seed, the module which grows horizontally and initiates growth of sets of modules to its north is called the `planter`. The `planter` grows in a zig-zag, up and down manner, growing one column at a time. Essentially, its job is to manage a set of nested counters, whose values are used to (1) determine the correct spacing between the modules to the `planter`'s north, and (2) serve as input to those modules. The outermost of the nested counters counts $0 < i < \infty$, with each i being what we call an *iteration*. For each value of i that it counts, it holds that counter constant while it increments an inner

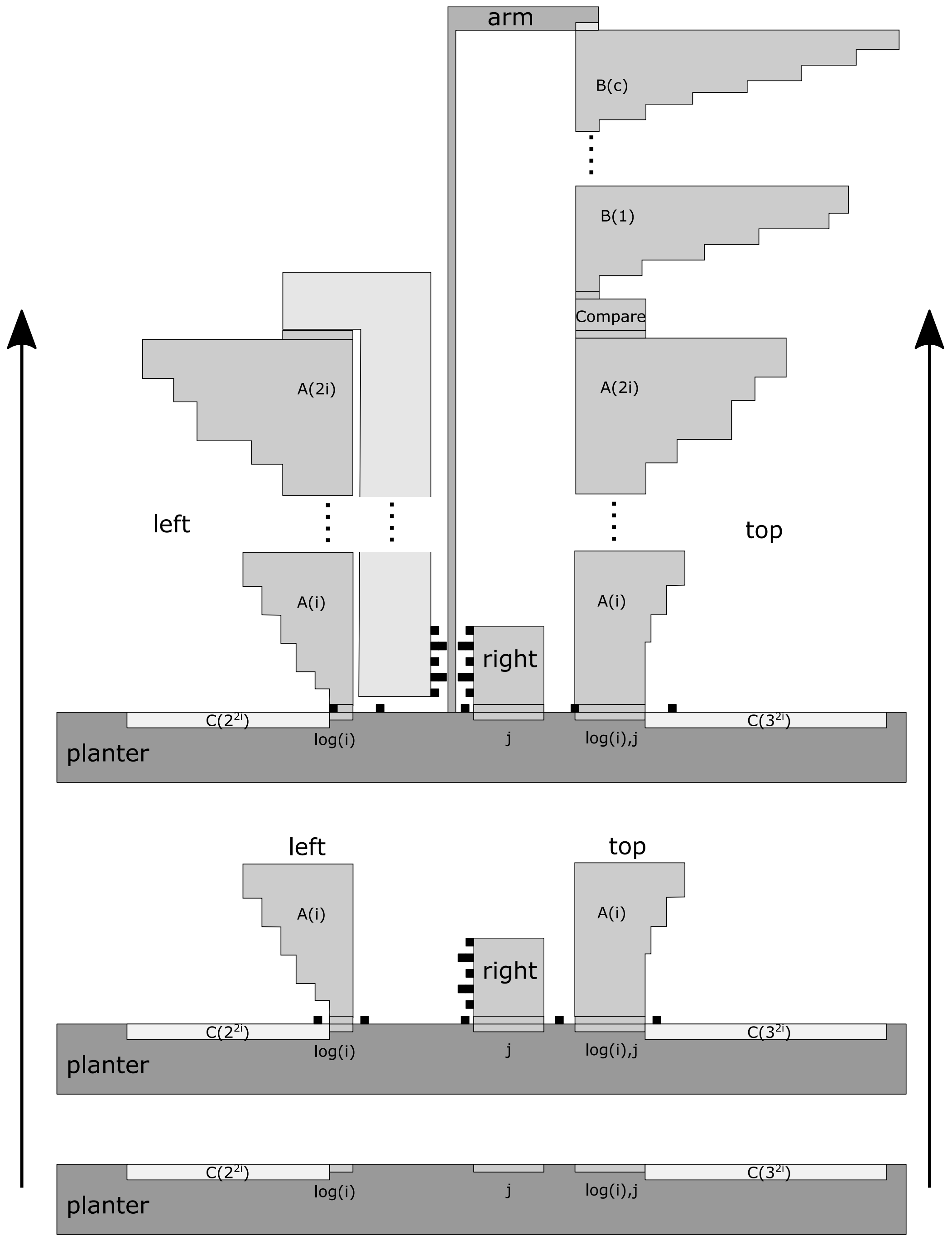

Figure 3: A high-level schematic depiction of one possible ordering of growth of the modules of an empty subiteration. (Bottom) The `planter` lays out the inputs for the modules at the necessary spacings to prevent them from colliding, (Second) The `left`, `right`, and `top` modules begin growth, (Top) Once the `top` completes it initiates the growth of the `arm` which grows down through the `bitAlley`. Note that an `arm` only grows in the `bitAlley` of an empty subiteration, unlike the `bitAlley` in Figure 4 which shows tiles cooperatively binding across the `bitAlley` of a non-empty subiteration. Also, empty subiterations occur exponentially more rarely than non-empty ones.

counter from 0 to (approximately) 2^i. For each value of j it initiates the growth of what we call a *subiteration*. See Figure 3 for a high-level overview of one type of subiteration. For each subiteration, the `planter` counts out a sequence of spacing columns (i.e. columns whose sole purpose is to put horizontal space between modules) while also computing the value $\log(i)$ and then rotating the values of the bits representing $\log(i)$ upward so that they are encoded in a row of glues on the north sides of the northern tiles of the `planter`[2]. From these, a `left` module begins growth. This module performs a stacked up series of i Turing machine simulations on progressively increasing input values, with each simulation outputting a 0 (for a rejecting computation) or a 1 (accepting). At the top of the stack of computations, the string of output bits is rotated to the right and then grown downward to the right of the `left` module. Once that growth reaches a specially marked location, the values of those bits are rotated to the right where they are presented as the eastern glues of the tiles forming the `bitAlley`. (See Figure 4 for a depiction of a southern portion of a `bitAlley`.)

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

The final module to be initiated by the `planter` in each subiteration is the `top` module. This module receives as input both the values $\log(i)$ and j. It first performs the same i simulations that the `left` performs, generating the same output bits. It then compares those bits to the bits of j to determine if there are any locations where the bits are the same. If there are, then in the `bitAlley` there will be tiles which attach between them across the gap in those locations, and the `top` module halts its growth (in this subiteration). It is guaranteed that in exactly one subiteration of each iteration that there will be no matching bits, since each subiteration performs the same `left` computations on the same input and there is a unique subiteration for every possible bit string of length i, exactly one of which can be the complement of `left`'s output on that input. In this special subiteration of the iteration, which we call the *empty* subiteration (because the `bitAlley` will be empty of tiles cooperating across the gap), the `top` performs a new set of computations to determine which of a large number

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

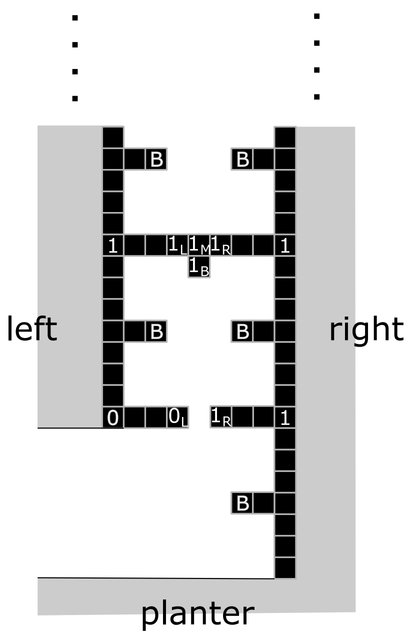

Figure 4: Example `bitAlley` portion between `left` and `right` modules of a non-empty subiteration.

(relative to the number of tile types in the claimed universal simulator U) of `arm` modules to grow. The `arm` module grows over to a position directly above the `bitAlley`, then grows a single tile wide column of tiles down through the `bitAlley` until it crashes into the `planter`, with the specific type of tile used for the `arm` determined by the final computations performed by the `top` module. This completes the growth of a subiteration, and the growth of subiterations and iterations occurs for infinite numbers of each.

B. *Directedness of $\mathcal{T}$*

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

VI. Overview of Impossibility of Simulation

In this section is a high-level overview of the proof that $\mathcal{S}$ does not simulate $\mathcal{T}$. More details can be found in [21].

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

A key tool in the proof is that in an arbitrary subiteration j of an arbitrary iteration i, the output of the `left` module is impossible to compute from within either the `planter` or the `right` modules, and the output of the `right` is impossible to compute from within the `left`. The reason for this is that (1) the Turing machines being simulated within the `left` modules are deciding languages which cannot be recognized in infinitely often best-case space complexity [20] which is greater than the space resources available to the `planter` and `right` modules, and thus the outputs of `left` modules cannot be computed by them, and (2) the input j passed to the `right` module is asymptotically much greater in size than the amount of information which can be input to the `left` module through the only $\log(\log(i))$ macrotiles allowed in the bottom row of the `left` module to encode the value $\log(i)$, making it unable to get asymptotically more than a log size chunk of the `right` module's input. It is also important to note the languages being decided within the `left` are recognized in almost everywhere worst case space complexity which is accounted for by the spacing columns of the `planter`, guaranteeing that for all but a finite number of computations, the `left` will be able to successfully complete its computations. It will prematurely abort any computations which attempt to run beyond those space bounds, but since there are guaranteed to be only a finite number of those, the goals of the construction and correctness of the proof aren't compromised. It is important that these essentially arbitrarily tight bounds on the space complexities of languages is shown to be possible by Theorem 4.1 of [20], which allows for the computations embedded within the modules to be designed with great precision. In a similar manner, the computations performed by the upper portion of the `top` module require space complexity greater that that available to either the `planter` or `left` of the same subiteration. We note that Lemma 9.14 (see [21]) is instrumental in proving the above facts, and is also an important tool which can be used in future simulation-based results in the aTAM, as it proves that an assembly performing a simulation of a system growing in a zig-zag manner, despite its arbitrarily large (but constant) scale factor, has asymptotically no greater space resources available than the orignal system. The technical tools we have developed for this proof, as well as the incorporation of results from complexity theory allowing for precisely defined languages in terms of space complexity, provide a host of new construction and proof techniques which we feel will be useful for a variety of future results.

To prevent the simulator from being seeded with answers to the necessary computations, the assembly of $\mathcal{T}$ must grow infinitely many iterations and subiterations. To prevent other types of "cheating", rather than having potential locations of cooperation across a single gap between two tiles, the `bitAlley` becomes arbitrarily long, between an arbitrarily large set of pairs of tiles. To prove all of the necessary properties of the simulator $\mathcal{S}$ requires many more details and the use of several additional technical lemmas which may possibly be of independent interest and utility. Please see [21] for full details.

Acknowledgements

The authors would like to thank Jack Lutz for helpful guidance while searching for much needed computational complexity results. This research was supported in part by National Science Foundation Grants CCF-1117672 and

CCF-1422152, and Graduate Research Fellowship Grant DGE-1450079.

References

[1] Y. Ke, L. L. Ong, W. M. Shih, and P. Yin, "Three-dimensional structures self-assembled from dna bricks," *Science*, vol. 338, no. 6111, pp. 1177–1183, 2012.

[2] P. W. Rothemund, N. Papadakis, and E. Winfree, "Algorithmic self-assembly of dna sierpinski triangles," *PLoS biology*, vol. 2, no. 12, p. e424, 2004.

[3] G. M. Whitesides and M. Boncheva, "Beyond molecules: Self-assembly of mesoscopic and macroscopic components," *Proceedings of the National Academy of Sciences*, vol. 99, no. 8, pp. 4769–4774, 2002. [Online]. Available: http://www.pnas.org/content/99/8/4769.abstract

[4] E. Winfree, "Algorithmic self-assembly of DNA," Ph.D. dissertation, California Institute of Technology, June 1998.

[5] D. Soloveichik and E. Winfree, "Complexity of self-assembled shapes," *SIAM Journal on Computing*, vol. 36, no. 6, pp. 1544–1569, 2007.

[6] P. W. K. Rothemund and E. Winfree, "The program-size complexity of self-assembled squares (extended abstract)," in *STOC '00: Proceedings of the thirty-second annual ACM Symposium on Theory of Computing*. Portland, Oregon, United States: ACM, 2000, pp. 459–468.

[7] D. Doty, J. H. Lutz, M. J. Patitz, R. T. Schweller, S. M. Summers, and D. Woods, "The tile assembly model is intrinsically universal," in *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science*, ser. FOCS 2012, 2012, pp. 302–310.

[8] E. D. Demaine, M. J. Patitz, T. A. Rogers, R. T. Schweller, S. M. Summers, and D. Woods, "The two-handed assembly model is not intrinsically universal," in *40th International Colloquium on Automata, Languages and Programming, ICALP 2013, Riga, Latvia, July 8-12, 2013*, ser. Lecture Notes in Computer Science. Springer, 2013.

[9] J. Hendricks, M. J. Patitz, T. A. Rogers, and S. M. Summers, "The power of duples (in self-assembly): It's not so hip to be square," in *Computing and Combinatorics - 20th International Conference, (COCOON) 2014, Atlanta, GA, USA, August 4-6, 2014. Proceedings*, 2014, pp. 215–226.

[10] T. Fochtman, J. Hendricks, J. E. Padilla, M. J. Patitz, and T. A. Rogers, "Signal transmission across tile assemblies: 3d static tiles simulate active self-assembly by 2d signal-passing tiles," *Natural Computing*, vol. 14, no. 2, pp. 251–264, 2015.

[11] J. Hendricks, M. J. Patitz, and T. A. Rogers, "Doubles and negatives are positive (in self-assembly)," *Natural Computing*, vol. 15, no. 1, pp. 69–85, 2016. [Online]. Available: http://dx.doi.org/10.1007/s11047-015-9513-6

[12] ——, "The simulation powers and limitations of higher temperature hierarchical self-assembly systems," in *7th International Conference on Machines, Computations and Universality (MCU'15), (9-11 September, 2015, Eastern Mediterranean University, Famagusta, North Cyprus)*, pp. 149–163.

[13] P.-E. Meunier, M. J. Patitz, S. M. Summers, G. Theyssier, A. Winslow, and D. Woods, "Intrinsic universality in tile self-assembly requires cooperation," in *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA 2014), (Portland, OR, USA, January 5-7, 2014)*, 2014, pp. 752–771.

[14] O. Gilber, J. Hendricks, M. J. Patitz, and T. A. Rogers, "Computing in continuous space with self-assembling polygonal tiles," in *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016), Arlington, VA, USA* January 10-12, 2016, pp. 937–956.

[15] S. P. Fekete, J. Hendricks, M. J. Patitz, T. A. Rogers, and R. T. Schweller, "Universal computation with arbitrary polyomino tiles in non-cooperative self-assembly," in *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015), San Diego, CA, USA* January 4-6, 2015, pp. 148–167. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/1.9781611973730.12

[16] E. D. Demaine, M. L. Demaine, S. P. Fekete, M. J. Patitz, R. T. Schweller, A. Winslow, and D. Woods, "One tile to rule them all: Simulating any tile assembly system with a single universal tile," in *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP 2014),* IT University of Copenhagen, Denmark, July 8-11, 2014, ser. LNCS, vol. 8572, 2014, pp. 368–379.

[17] N. Bryans, E. Chiniforooshan, D. Doty, L. Kari, and S. Seki, "The power of nondeterminism in self-assembly," in *SODA 2011: Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2011, pp. 590–602.

[18] D. Doty, "Randomized self-assembly for exact shapes," in *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2009, pp. 85–94.

[19] M.-Y. Kao and R. T. Schweller, "Randomized self-assembly for approximate shapes," in *ICALP (1)*, ser. Lecture Notes in Computer Science, L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, Eds., vol. 5125. Springer, 2008, pp. 370–384.

[20] J. D. P. Rolim and S. A. Greibach, "A note on the best-case complexity," *Inf. Process. Lett.*, vol. 30, no. 3, pp. 133–138, 1989. [Online]. Available: http://dx.doi.org/10.1016/0020-0190(89)90131-2

[21] J. Hendricks, M. J. Patitz, and T. A. Rogers, "Universal simulation of directed systems in the abstract tile assembly model requires undirectedness," Computing Research Repository, Tech. Rep. 1608.03036, 2016. [Online]. Available: http://arxiv.org/abs/1608.03036

[22] M. J. Patitz, R. T. Schweller, and S. M. Summers, "Exact shapes and turing universality at temperature 1 with a single negative glue," in *Proceedings of the 17th international conference on DNA computing and molecular programming*, ser. DNA'11, 2011, pp. 175–189.

[23] M. J. Patitz and S. M. Summers, "Self-assembly of decidable sets," *Natural Computing*, vol. 10, no. 2, pp. 853–877, 2011.

[24] J. I. Lathrop, J. H. Lutz, M. J. Patitz, and S. M. Summers, "Computability and complexity in self-assembly," *Theory Comput. Syst.*, vol. 48, no. 3, pp. 617–647, 2011.