Computability and Complexity in Self-Assembly

James I. Lathrop*, Jack H. Lutz†‡, Matthew J. Patitz§, and Scott M. Summers¶‖

Abstract

This paper explores the impact of geometry on computability and complexity in Winfree's model of nanoscale self-assembly. We work in the two-dimensional tile assembly model, i.e., in the discrete Euclidean plane $\mathbb{Z} \times \mathbb{Z}$. Our first main theorem says that there is a roughly quadratic function f such that a set $A \subseteq \mathbb{Z}^+$ is computably enumerable if and only if the set $X_A = \{(f(n), 0) \mid n \in A\}$ – a simple representation of A as a set of points on the x-axis – self-assembles in Winfree's sense. In contrast, our second main theorem says that there are decidable sets $D \subseteq \mathbb{Z} \times \mathbb{Z}$ that do *not* self-assemble in Winfree's sense.

Our first main theorem is established by an explicit translation of an arbitrary Turing machine M to a modular tile assembly system $\mathcal{T}_M$, together with a proof that $\mathcal{T}_M$ carries out concurrent simulations of M on all positive integer inputs.

1 Introduction

A major objective of nanotechnology is to engineer systems that, like many systems in nature, autonomously assemble themselves from molecular components. One promising approach to this objective, pioneered by Seeman in the 1980s [11], is *DNA tile self-assembly*. This approach exploits the information-processing and structural properties of DNA to construct nanoscale components (DNA tiles) that, with moderate control of physical conditions (temperature, concentration in solution, etc.), spontaneously assemble themselves into larger structures. These structures, which may be complex and aperiodic, are *programmed*, in the sense that they are determined by a designer's selection of the short, single-strand nucleotide sequences ("sticky ends") that appear on each side of each type of DNA tile to be used.

In the late 1990s, Winfree [15] introduced a mathematical model of DNA tile assembly called the Tile Assembly Mode. This model, which was soon refined by Rothemund and Winfree [10, 9], is an extension of Wang tiling [13, 14]. (see also [1, 8, 12].) The Tile Assembly Model, which is described in section 2 below, uses square tiles with various types and strengths of "glue" on their edges as abstractions of DNA tiles adsorbing to a growing structure on a very flat surface such as mica. Winfree [15] proved that the Tile Assembly Model is computationally universal, i.e., that any Turing machine can be encoded into a finite set of tile types whose self-assembly simulates that Turing machine.

The computational universality of the Tile Assembly Model implies that self-assembly can be algorithmically directed, and hence that a very rich set of structures can be formed by self-assembly. However, as we shall see, this computational universality does not seem to imply a simple characterization of the class of structures that can be formed by self-assembly. The difficulty is that self-assembly (like sensor networks, smart materials, and other topics of current research [2, 3]) is a phenomenon in which the *spatial* aspect of

*Department of Computer Science, Iowa State University, Ames, IA 50011, U.S.A. Email: jil@cs.iastate.edu

†Department of Computer Science, Iowa State University, Ames, IA 50011, U.S.A. Email: lutz@cs.iastate.edu

‡This author's research was supported in part by National Science Foundation Grants 0344187, 0652569 and 0728806 and by Spanish Government MEC Project TIN 2005-08832-C03-02

§Department of Computer Science, Iowa State University, Ames, IA 50011, U.S.A. Email: mpatitz@cs.iastate.edu

¶Department of Computer Science, Iowa State University, Ames, IA 50011, U.S.A. Email: summers@cs.iastate.edu

‖This author's research was supported in part by NSF-IGERT Training Project in Computational Molecular Biology Grant number DGE-0504304

computing plays a crucial role. Two processes, namely self-assembly and the computation that directs it, must take place in the same space and time.

In this paper, we focus on the self-assembly that takes place in the discrete Euclidean plane $\mathbb{Z}^2 = \mathbb{Z} \times \mathbb{Z}$, where $\mathbb{Z}$ is the set of integers. Roughly, a *tile assembly system* $\mathcal{T}$ (defined precisely in section 2) is a finite set T of tile types, together with a finite seed assembly σ consisting of one or more tiles of these types. Self-assembly is then the process in which tiles of the types in T successively adsorb to σ (more precisely, to the assembly which is thereby dynamically growing from σ) in any manner consistent with the rules governing the glue types on the edges of the given tile types. Note that self-assembly is, in general, a nondeterministic, asynchronous process.

We say that a set $X \subseteq \mathbb{Z}^2$ *self-assembles* in a tile assembly system $\mathcal{T}$ if every possible "run" of self-assembly in $\mathcal{T}$ results in the placement of black tiles on the set X and only on the set X. (Some of the tile types in $\mathcal{T}$ are designated to be black. Non-black tiles may appear on some or all points in the complement $\mathbb{Z}^2 - X$.) This is the sense in which Winfree [15] has demonstrated the self-assembly of the standard discrete Sierpinski triangle. (In [7] this is called *weak self-assembly* to contrast it with a stricter notion in which, essentially, all tiles are required to be black.)

This paper presents two main theorems on the interplay between geometry and computation in tile self-assembly. To explain our first main theorem, define the function $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ by

$$f(n) = \binom{n+1}{2} + (n+1)\lfloor \log n \rfloor + 6n - 2^{1+\lfloor \log(n) \rfloor} + 2.$$

Note that f is a reasonably simple, strictly increasing, roughly quadratic function of n. For each set $A \subseteq \mathbb{Z}^+$, the set

$$X_A = \{(f(n), 0) | n \in A\}$$

is thus a straightforward representation of A as a set of points on the positive x-axis.

Our first main theorem says that *every* computably enumerable set A of positive integers (decidable or otherwise) self-assembles in the sense that there is a tile assembly system $\mathcal{T}_A$ in which the representation X_A self-assembles. Conversely, the existence of such a tile assembly system implies the computable enumerability of A.

In contrast, our second main theorem says that there are *decidable* sets $D \subseteq \mathbb{Z}^2$ that do *not* self-assemble in any tile assembly system. In fact, we exhibit such a set D for which the condition $(m, n) \in D$ is decidable in time polynomial in $|m| + |n|$.

Taken together, our two main theorems indicate that the interaction between geometry and computation in self-assembly is not at all simple. Further investigation of this interaction will improve our understanding of tile self-assembly and, more generally, spatial computation.

The proof of our first main theorem has two features that may be useful in future investigations. First, we give an explicit transformation (essentially a compiler, implemented in C++) of an arbitrary Turing machine M to a tile assembly system $\mathcal{T}_M$ whose self-assembly carries out concurrent simulations of M on (the binary representation of) all positive integer inputs. Second, we prove two lemmas – a pseudoseed lemma and a multiseed lemma – that enable us to reason about tile assembly systems in a modular fashion. This modularity, together with the local determinism method of Soloveichik and Winfree [12], enables us to prove the correctness of $\mathcal{T}_M$.

Images are in black and white in the print version of this paper and in color in the online version.

2 Preliminaries

We work in the 2-dimensional discrete Euclidean space $\mathbb{Z}^2$. We write U_2 for the set of all *unit vectors*, i.e., vectors of length 1, in $\mathbb{Z}^2$. We regard the 4 elements of U_2 as (names of the cardinal) *directions* in $\mathbb{Z}^2$.

We now give a brief and intuitive sketch of the Tile Assembly Model that is adequate for reading this paper. More formal details and discussion may be found in [15, 10, 9, 7]. Our notation is that of [7].

Intuitively, a tile type t is a unit square that can be translated, but not rotated, so it has a well-defined "side $\vec{u}$" for each $\vec{u} \in U_2$. Each side $\vec{u}$ is covered with a "glue" of "color" $\text{col}_t(\vec{u})$ and "strength" $\text{str}_t(\vec{u})$ specified by its type t. If two tiles are placed with their centers at adjacent points $\vec{m}, \vec{m} + \vec{u} \in \mathbb{Z}^2$, where $\vec{u} \in U_2$, and if their abutting sides have glues that match in both color and strength, then they form a *bond* with this common strength. If the glues do not so match, then no bond is formed between these tiles. In this paper, all glues have strength 0, 1, or 2. When drawing a tile as a square, each side's glue strength is indicated by whether the side is a dotted line (0), a solid line (1), or a double line (2). Each side's "color" is indicated by an alphanumeric label.

Given a set T of tile types and a "temperature" $\tau \in \mathbb{N}$, a *τ-T-assembly* is a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$ - intuitively, a placement of tiles at some locations in $\mathbb{Z}^2$ - that is *τ-stable* in the sense that it cannot be "broken" into smaller assemblies without breaking bonds of total strength at least τ. For each $t \in T$, we write $\partial_t \alpha$ for the set of all locations to which a tile of type t can be τ-stably added. We write $\partial\alpha$ for the set of all locations to which some tile can be τ-stably added to α. We refer to $\partial\alpha$ as the *frontier* of α. If α and α' are assemblies, then α is a *subassembly* of α', and we write $\alpha \sqsubseteq \alpha'$, if $\text{dom}\ \alpha \subseteq \text{dom}\ \alpha'$ and $\alpha(\vec{m}) = \alpha'(\vec{m})$ for all $\vec{m} \in \text{dom}\ \alpha$.

Self-assembly begins with a *seed assembly* σ and proceeds asynchronously and nondeterministically, with tiles adsorbing one at a time to the existing assembly in any manner that preserves τ-stability at all times. A *tile assembly system* (*TAS*) is an ordered triple $\mathcal{T} = (T, \sigma, \tau)$, where T is a finite set of tile types, σ is a seed assembly with finite domain, and $\tau \in \mathbb{N}$. In this paper we always have $\tau = 2$. A *generalized tile assembly system* (*GTAS*) is defined similarly, but without the finiteness requirements. We write $\mathcal{A}[\mathcal{T}]$ for the set of all assemblies that can arise (in finitely many steps or in the limit) from $\mathcal{T}$. An assembly α is *terminal*, and we write $\alpha \in \mathcal{A}_\square[\mathcal{T}]$, if no tile can be τ-stably added to it.

An assembly sequence in a TAS $\mathcal{T}$ is a (finite or infinite) sequence $\vec{\alpha} = (\alpha_0, \alpha_1, \ldots)$ of assemblies in which each α_{i+1} is obtained from α_i by the addition of a single tile. The *result* $\text{res}(\vec{\alpha})$ of such an assembly sequence is its unique limiting assembly (This is the last assembly in the sequence if the sequence is finite). The set $\mathcal{A}[T]$ is partially ordered by the relation $\longrightarrow$ defined by

$$\begin{aligned} \alpha \longrightarrow \alpha' \quad \text{iff} \quad & \text{there is an assembly} \\ & \text{sequence } \vec{\alpha} = (\alpha_0, \alpha_1, \ldots) \\ & \text{such that } \alpha_0 = \alpha \text{ and} \\ & \alpha' = \text{res}(\vec{\alpha}). \end{aligned}$$

We say that $\mathcal{T}$ is *directed* if the relation $\longrightarrow$ is directed, i.e., if for all $\alpha, \alpha' \in \mathcal{A}[T]$, there exists $\alpha'' \in \mathcal{A}[T]$ such that $\alpha \longrightarrow \alpha''$ and $\alpha' \longrightarrow \alpha''$. It is easy to show that $\mathcal{T}$ is directed if and only if there is a unique terminal assembly $\alpha \in \mathcal{A}[T]$ such that $\sigma \longrightarrow \alpha$.

In general, even a directed TAS may have a very large (perhaps uncountably infinite) number of different assembly sequences leading to its terminal assembly. This seems to make it very difficult to prove that a TAS is directed. Fortunately, Soloveichik and Winfree [12] have recently defined a property, *local determinism*, of assembly sequences and proven the remarkable fact that, if a TAS $\mathcal{T}$ has *any* assembly sequence that is locally deterministic, then $\mathcal{T}$ is directed.

A set $X \subseteq \mathbb{Z}^2$ *(weakly) self-assembles* if there exist a TAS $\mathcal{T} = (T, \sigma, \tau)$ and a set $B \subseteq T$ such that $\alpha^{-1}(B) = X$ holds for every terminal assembly $\alpha \in \mathcal{A}_\square[T]$.

3 Pseudoseeds and Multiseeds

This section introduces two conceptual tools that enable us to reason about tile assembly systems in a modular fashion.

The idea of our first tool is intuitive. Suppose that our objective is to design a tile assembly system $\mathcal{T}$ in which a given set $X \subseteq \mathbb{Z}^2$ self-assembles. The set X might have a subset X^* for which it is natural to decompose the design into the following two stages.

(i) Design a TAS $\mathcal{T}_0 = (T_0, \sigma, \tau)$ in which an assembly σ^* with domain X^* self-assembles.

(ii) Extend T_0 to a tile set T such that X self-assembles in the TAS $\mathcal{T}^* = (T, \sigma^*, \tau)$

We would then like to conclude that X self-assembles in the TAS $\mathcal{T} = (T, \sigma, \tau)$. This will not hold in general, but it does hold if (i) continues to hold with T in place of T_0 and σ^* is a pseudoseed in the following sense.

Definition. Let $\mathcal{T} = (T, \sigma, \tau)$ be a GTAS. A *pseudoseed* of $\mathcal{T}$ is an assembly $\sigma^* \in \mathcal{A}[\mathcal{T}]$ with the property that, if we let $\mathcal{T}^* = (T, \sigma^*, \tau)$, then, for every assembly $\alpha \in \mathcal{A}[\mathcal{T}]$, there exists an assembly $\alpha' \in \mathcal{A}[\mathcal{T}^*]$ such that $\alpha \sqsubseteq \alpha'$.

The following lemma says that the above definition has the desired effect.

Lemma 3.1 (pseudoseed lemma)**.** If σ^* is a pseudoseed of a GTAS $\mathcal{T} = (T, \sigma, \tau)$ and $\mathcal{T}^* = (T, \sigma^*, \tau)$, then $\mathcal{A}_\square[\mathcal{T}] = \mathcal{A}_\square[\mathcal{T}^*]$.

Proof. ("$\subseteq$" direction) Let $\alpha \in \mathcal{A}_\square[\mathcal{T}]$. This means that $\sigma \to \alpha$. The definition of pseudoseed tells us that there exists $\alpha' \in \mathcal{A}[\mathcal{T}^*]$ such that $\alpha \sqsubseteq \alpha'$. But since α is terminal, we must have $\alpha = \alpha'$, whence $\sigma \sqsubseteq \sigma^* \sqsubseteq \alpha$. It follows, by Rothemund's lemma (Lemma 2 on p. 57 of [9]), that $\sigma^* \to \alpha$.

("$\supseteq$" direction) Let $\alpha \in \mathcal{A}_\square[\mathcal{T}^*]$. Then we have $\sigma^* \to \alpha$. Moreover, $\sigma \to \sigma^*$ because $\sigma^* \in \mathcal{A}[\mathcal{T}]$. Thus, $\sigma \to \sigma^* \to \alpha$, and it follows, by the transitivity of $\to$, that $\alpha \in \mathcal{A}_\square[\mathcal{T}]$. □

Note that the pseudoseed lemma entitles us to *reason as though* the self-assembly proceeds in stages, even though this may not actually occur. (E.g., the pseudoseed σ^* may itself be infinite, in which case the self-assembly *of* σ^* and the self-assembly *from* σ^* must occur concurrently.)

Our second tool for modular reasoning is a bit more involved. Suppose that we have a tile set T and list $\sigma_0, \sigma_1, \sigma_2, \ldots$ of seeds that, for each i, the TAS $\mathcal{T}_i = (T, \sigma_i, \tau)$ has a desired assembly α_i as its result. If the assemblies $\alpha_0, \alpha_1, \alpha_2, \ldots$ have disjoint domains, then it might be possible for *all* these assemblies to grow from a "multiseed" σ^* that has $\sigma_0, \sigma_1, \sigma_2, \ldots$ embedded in it. We now define a sufficient condition for this.

Definition. Let T and T' be sets of tile types with $T \subseteq T'$, and let $\vec{\sigma} = (\sigma_i \mid 0 \leq i < k)$ be a sequence of τ-T-assemblies, where $k \in \mathbb{Z}^+ \cup \{\infty\}$. A $\vec{\sigma}$-T-T'-*multiseed* is a τ-T' assembly σ^* with the property that, if we write

$$\mathcal{T}^* = (T', \sigma^*, \tau)$$

and

$$\mathcal{T}_i = (T, \sigma_i, \tau)$$

for each $0 \leq i < k$, then the following four conditions hold.

1. For each i, $\sigma_i \sqsubseteq \sigma^*$.
2. For each $i \neq j$, $\alpha \in \mathcal{A}[\mathcal{T}_i]$, $\alpha' \in \mathcal{A}[\mathcal{T}_j]$, $\vec{m} \in \text{dom } \alpha$, and $\vec{m'} \in \text{dom } \alpha'$, $\vec{m} - \vec{m'} \in U_2 \cup \{\vec{l}\} \Rightarrow \vec{m}, \vec{m'} \in \text{dom } \sigma^*$. (Recall that U_2 is the set of unit vectors in $\mathcal{Z}^2$.)
3. For each i, and each $\alpha \in \mathcal{A}[\mathcal{T}_i]$, there exists $\alpha^* \in \mathcal{A}[\mathcal{T}^*]$ such that $\alpha \sqsubseteq \alpha^*$.
4. For each $\alpha^* \in \mathcal{A}[\mathcal{T}^*]$, there exists, for each $0 \leq i < k$, $\alpha_i \in \mathcal{A}[\mathcal{T}_i]$ such that $\alpha^* \sqsubseteq \sigma^* + \sum_{0 \leq i < k} \alpha_i$.
5. For each $\alpha \in \mathcal{A}[\mathcal{T}^*]$, $\alpha^{-1}(T' - T) \subseteq \text{dom } \sigma^*$.

Note: In condition 4 we are using the operation + defined as follows. If $\alpha, \alpha' : \mathbb{Z}^2 \dashrightarrow T$ are *consistent*, in the sense that they agree on $\text{dom } \alpha \cap \text{dom } \alpha'$, then $\alpha + \alpha' : \mathbb{Z}^2 \dashrightarrow T$ is the unique partial function satisfying $\text{dom } (\alpha + \alpha') = \text{dom } \alpha \cup \text{dom } \alpha'$, $\alpha \sqsubseteq \alpha + \alpha'$, and $\alpha' \sqsubseteq \alpha + \alpha'$. This is extended to summations $\sum_{0 \leq i < k} \alpha_i$ in the obvious way. The assemblies being summed in condition 4 are consistent by conditions 1 and 2.

Intuitively, the four conditions in the above definition can be stated as follows.

1. The seeds σ_i are embedded in σ^*.
2. Assemblies in $\mathcal{A}[\mathcal{T}_i]$ and assemblies in $\mathcal{A}[\mathcal{T}_j]$ do not interfere with each other.
3. σ^* does not interfere with assemblies in $\mathcal{A}[\mathcal{T}_i]$.
4. σ^* does not produce anything other than what its embedded seeds σ_i produce.
5. Tile types in $T' - T$ cannot occur outside σ^*.

The following lemma says that the multiseed definition has the desired effect.

Lemma 3.2 (multiseed lemma)**.** Let $T \subseteq T'$ be sets of tile types, and let $\vec{\sigma} = (\sigma_i \mid 0 \leq i < k)$ be a sequence of τ-T-assemblies, where $k \in \mathbb{Z}^+ \cup \{\infty\}$. If σ^* is a $\vec{\sigma}$-T-T'-multiseed of $\mathcal{T}^*$ and $\mathcal{T}_i$ $(0 \leq i < k)$ are defined as in the multiseed definition, then

$$\mathcal{A}_{\square}[\mathcal{T}^*] = \left\{ \sigma^* + \sum_{0 \leq i < k} \alpha_i \;\middle|\; \text{each } \alpha_i \in \mathcal{A}_{\square}[\mathcal{T}_i] \right\}.$$

Proof. ("$\subseteq$" direction) Let $\alpha^* \in \mathcal{A}[\mathcal{T}^*]$. It follows by part (4) of definition 3 that $\alpha^* = \sigma^* + \sum_{0 \leq i < k} \alpha_i$, where $\alpha_i \in \mathcal{A}[\mathcal{T}_i]$, Since α^* is terminal, for each i, $\alpha_i \in \mathcal{A}_{\square}[\mathcal{T}_i]$, whence $\alpha^* \in \left\{ \sigma^* + \sum_{0 \leq i < k} \alpha_i \;\middle|\; \text{each } \alpha_i \in \mathcal{A}_{\square}[\mathcal{T}_i] \right\}$.

("$\supseteq$" direction) For each $0 \leq i < k$, let $\alpha_i \in \mathcal{A}_{\square}[\mathcal{T}_i]$ and let $\alpha = \sigma^* + \sum_{0 \leq i < k} \alpha_i$. It suffices to shaw that $\alpha \in \mathcal{A}_{\square}[\mathcal{T}^*]$.

Condition 3 of the multiseed definition tells us that, for each $0 \leq i < k$, there is an assembly $\alpha_i^* \in \mathcal{A}[\mathcal{T}^*]$ such that $\alpha_i \sqsubseteq \alpha_i^*$. Since $\sigma^* \sqsubseteq \alpha_i^*$, it follows that

$$\sigma^* + \alpha_i \sqsubseteq \alpha_i^*.$$

By condition 4 of the multiseed definition, there is, for each $0 \leq i < k$ and $0 \leq j < k$, an assembly $\alpha_{ij} \in \mathcal{A}[\mathcal{T}_i]$ such that, for all $0 \leq i < k$,

$$\alpha_i^* \sqsubseteq \sigma^* + \sum_{0 \leq j < k} \alpha_{ij}.$$

For each $0 \leq i < k$, let

$$D_i = \text{dom } \alpha_i^* \cap (\text{dom } \sigma^* \cup \text{dom } \alpha_{ii}),$$

and let

$$\hat{\alpha}_i^* = \alpha_i^* \upharpoonright D_i.$$

By (1), (2), and condition 2 of the multiseed definition, we have

$$\sigma^* + \alpha_i \sqsubseteq \hat{\alpha}_i^*$$

for all $0 \leq i < k$. Now the assemblies $\hat{\alpha}_i^*$ are all consistent with one another, and each α_i is terminal in $\mathcal{A}[\mathcal{T}_i]$, so, by condition 5 of the multiseed definition, we must have

$$\alpha = \sum_{0 \leq i < k} \hat{\alpha}_i^*.$$

For each $0 \leq i < k$, let $\vec{\alpha}^{*i}$ be an assembly sequence from σ^* to α_i^*, and let $\vec{\hat{\alpha}}^{*i}$ be the sequence obtained from $\vec{\alpha}^{*i}$ by deleting all additions of tiles not in $\sigma^* + \alpha_{ii}$. By condition 2 of the multiseed definition, each $\vec{\hat{\alpha}}^{*i}$ is a τ-T'-assembly sequence from σ^* to $\hat{\alpha}_i^*$. By (3), then, if we dovetail the assembly sequences $\hat{\vec{\alpha}}^{*i} (0 \leq i < k)$, we get an assembly sequence $\hat{\vec{\alpha}}^*$ from σ^* to α, whence $\alpha \in \mathcal{A}[\mathcal{T}^*]$. Since the α_i are terminal, conditions 4 and 5 of the multiseed definition tell us that $\alpha \in \mathcal{A}_{\square}[\mathcal{T}^*]$. □

4 Self-Assembly of Computably Enumerable Sets

In [15], Winfree proved that the Tile Assembly Model is Turing universal in two dimensions. In this section, we prove a stronger result: for every TM M, there exists a directed TAS that simulates M on (the binary representation of) *every* input $x \in \mathbb{N}$ in the two dimensional discrete Euclidean plane. We state our result precisely in the following theorem.

Theorem 4.1 (first main theorem)**.** If $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ is defined as in section 1, then, for all $A \subseteq \mathbb{Z}^+$, A is computably enumerable if and only if the set $X_A = \{(f(n), 0) \mid n \in A\}$ self-assembles.

"$\Leftarrow$" direction. Let f be as defined in the introduction, and assume the hypothesis. Then there exists a 2-TAS $\mathcal{T} = (T, \sigma, \tau)$ in which the set X_A self-assembles. Let B be the set of "black" tile types given by the definition of self-assembly. Fix some enumeration $\vec{a}_1, \vec{a}_2, \vec{a}_3 \ldots$ of $\mathbb{Z}^2$, and let M be the TM, defined as follows.

Require: $n \in \mathbb{N}$
  $\alpha := \sigma$
  while $(f(n), 0) \notin \text{dom } \alpha$ **do**
    choose the least $j \in \mathbb{N}$ such that $\vec{a}_j \in \partial^\tau \alpha_i$
    choose $t \in T$ such that $\vec{a}_j \in \partial^\tau_t \alpha_i$
    $\alpha := \alpha + (\vec{a}_j \mapsto t)$
  end while
  if $\alpha((f(n), 0)) \in B$ **then**
    accept
  else
    reject
  end if

It is clear from above that $L(M) = A$, whence A is computably enumerable. □

To prove the "$\Rightarrow$" direction, we exhibit, for any TM M, a directed TAS $\mathcal{T}_M = (T, \sigma, \tau)$ that correctly simulates M on all inputs $x \in \mathbb{Z}^+$ in the two dimensional discrete Euclidean plane. We describe our construction, and give the complete specification for T in the remainder of this section.

4.1 Overview of Construction

Intuitively, $\mathcal{T}_M$ self-assembles a "gradually thickening bar", immediately below the positive x-axis with upward growths emanating from well-defined intervals of points. For each $x \in \mathbb{Z}^+$, there is an upward growth that simulates M on x. If M halts on x, then (a portion of) the upward growth associated with the simulation of $M(x)$ eventually stops, and sends a signal down along the right side of the upward growth via a one-tile-wide-path of tiles to the point $(f(x), 0)$, where a black tile is placed. See Figure 1 for a finite, yet intuitive snapshot of this infinite process.

Our tile assembly system $\mathcal{T}_M$ is divided into three modules: the ray, the planter, and the TM module, which control the spacing between successive simulations, the initiation of upward growth, and the actual simulations of M on each positive integer, respectively.

4.2 Overview Of The Ray Module

The first module in our construction is the ray module (middle shade of gray squares in Figure 1). For any $3 \leq w \in \mathbb{Z}^+$, a ray of width w is a fixed-width, periodic, binary counter that repeatedly counts from 0 to $2^w - 1$, such that each integer is counted once, and then immediately copied once before the value of the binary counter is incremented. Essentially, a ray of width w is a discrete line of constant thickness w, having a kind of "slope" that depends on w in the following way. In every other row (except for two special cases), the first tile to attach does so on top of the second-to-leftmost tile in the previous row. Thus, a ray of width

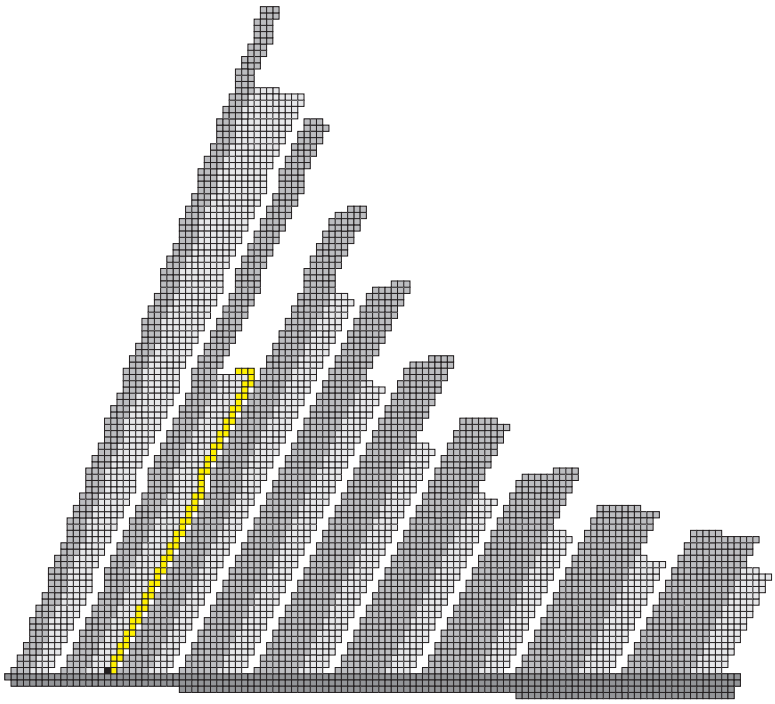


Figure 1: Simulation of M on every input $x \in \mathbb{N}$. Notice that $M(2)$ halts - indicated by the black tile along the x-axis.

w will have a slope of $\frac{2^w}{2^{w-1}-1} = 2 + \frac{2}{2^{w-1}-1}$. This implies that the set of points occupied by properly spaced, consecutive rays of strictly increasing width, will not only be disjoint but the width of the gap in between such rays will increase without limit.

4.3 Details Of Construction Of The Ray Module

(It is to be understood that, although not explicit in our discussion, tile types in each module are colored with a "module indicator" symbol to prevent erroneous binding between the tile types of different modules.)

The ray module is a tile set of 68 tile types. Although in our construction the ray is not a self-contained tile assembly system, it can be made to be one with a trivial change. For any $3 \leq w \in \mathbb{N}$, a ray of width w is a fixed-width, periodic, binary counter that repeatedly counts from 0 to $2^w - 1$, such that each integer is counted once, and then immediately copied once before the value of the binary counter is incremented. Our ray is based on the binary counter from [10], and thus increments right-to-left, and then copies the previous value left-to-right. However, since we construct the ray to operate on the reverse of each integer (i.e., the LSB of each integer is its left most bit), increments proceed left-to-right and copies right-to-left. The most important feature of the ray is that the first, and hence leftmost, tile to attach in any increment row does so on top of the second-to-leftmost tile in the previous copy row, *unless* the bit pattern of the previous increment row is of the form $1^{w-1}(0+1)$. A detailed example of the former case is shown in Figure 2.

In the latter case, the ray simply proceeds without "shifting" to the right by one unit. Copy rows are able to search for the two special bit patterns by using 'a' and 's' signals, respectively. For instance, when a copy row begins to self-assemble, and the current value of the counter is less than 2^{w-1}, an 'a' signal is propagated left through the copy row until a 0 bit is encountered in the previous increment row, thus breaking the signal. If no such bit exists, the signal will reach the leftmost tile of the copy row, and cause the next increment row to attach without being shifted one unit to the right. This situation is shown with

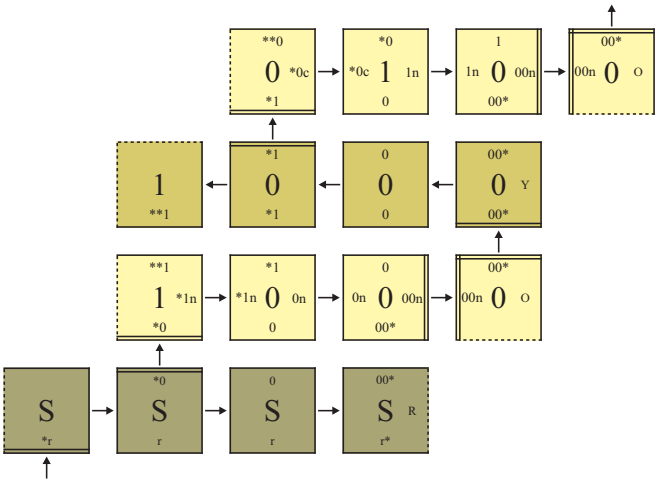


Figure 2: The first row of tiles is the initial row in a ray of width 4. The second row of tiles represents the value 1, and is the first increment row. The third row of tiles simply copies the value of the previous increment row.

detail in Figure 3.

Otherwise, the next increment row attaches on top of the second-to-leftmost tile in the current copy row. The 's' signal searches for the bit pattern 1^w in a similar fashion, shown with detail in Figure 4.

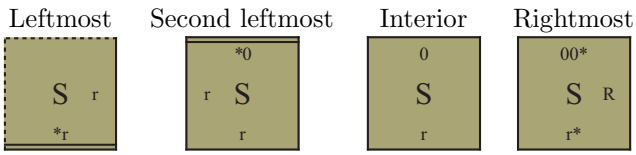
Finally, each row in the ray exhibits a particular "color" on the right side of its rightmost tile so as to allow a third (soon to be discussed) module to "know what to do and when to do it." Each increment row signals orange unless it contains the bit pattern $0^{w-1}(0+1)$, in which case it signals green for $0^{w-1}1$, and indigo otherwise. Each copy row signals yellow. However, if it is the copy row after an increment row with the bit pattern $0^{w-1}1$, then it signals blue. The initial row signals red.

In our construction, the upward growth of every ray is initiated by the planter. This can be seen with detail in Figure 5, where the top row of tiles (excluding the leftmost tile) will initiate the self-assembly of a ray having a width of 6. It is clear that our construction has the following properties.

1. A ray having a width of w will have a "slope" of $\frac{2^w}{2^{w-1}-1} = 2 + \frac{2}{2^{w-1}-1}$, which clearly tends to 2 as $w \to \infty$;
2. for every $3 \leq w \in \mathbb{Z}^+$, there is one and only one ray having a width of w;
3. thinner rays appear before thicker rays;
4. the set of points occupied by any ray is disjoint from the set of points occupied by any other ray and
5. the width between successive rays grows without limit.

The following is a list of tile types that make up the ray module.

1. Initial row - add the following tile types:



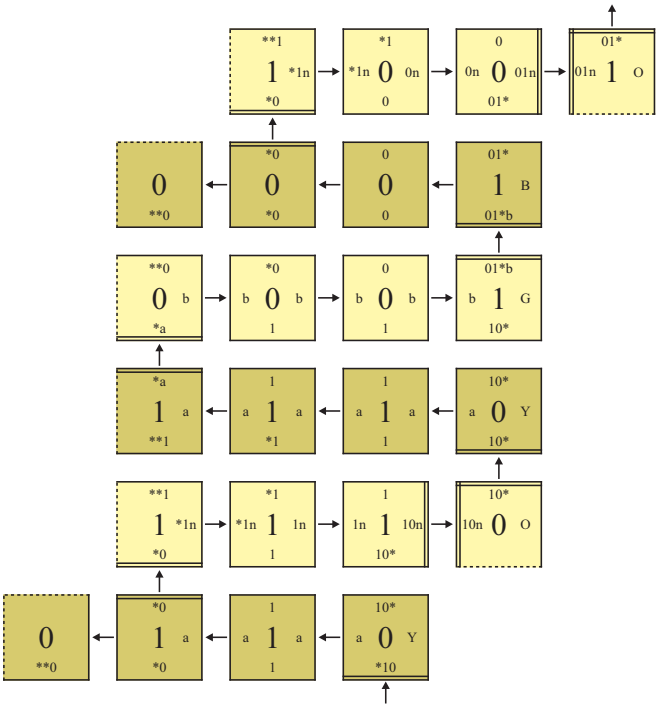
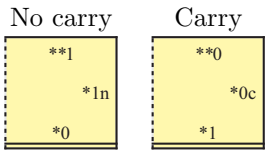


Figure 3: If the current value of the binary counter is a power of 2 (the fourth row of tiles), then the increment row that represents this value does not shift to the right by one unit. This situation is detected in the third (copy) row by the 'a' signal, which travels unimpeded right-to-left. The complementary 'b' signal is then sent left-to-right in the subsequent increment row to initiate a green signal.

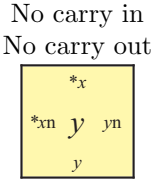
2. Tile types that perform general case increments (left to right).

   (a) Leftmost tile - add the following tile types:



   (b) Second leftmost tile type.

      i. If there is no carry bit to propagate, then for all $x, y \in \{0, 1\}$, add the following tile types:



      ii. If there is a carry bit coming in from the left, add the following tile types:

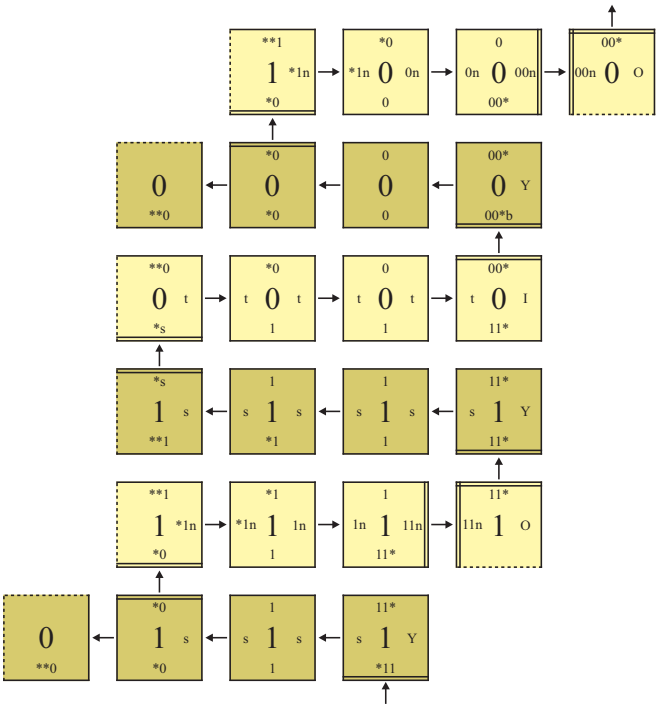
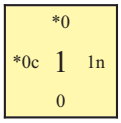
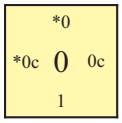


Figure 4: If the current value of the binary counter rolls over to all zeros (the fourth row of tiles), then the increment row that represents this value does not shift to the right by one unit. Similar to how the 'a' signal detects powers of 2, the 's' signal (the third row of tiles) detects when the counter is about to roll over. The 't' signal initiates an indigo signal.

Carry in, No carry out



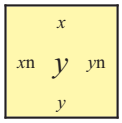
Carry in, Carry out



(c) Interior tile type.

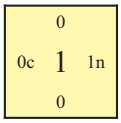
i. If there is no carry bit to propagate, then for all $x, y \in \{0, 1\}$, add the following tile types:

No carry in
No carry out

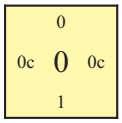


ii. If there is a carry bit coming in from the left, add the following tile types:

Carry in, No carry out



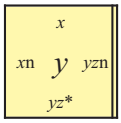
Carry in, Carry out



(d) Second rightmost tile type.

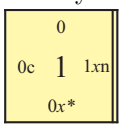
i. If there is no carry bit to propagate, then for all $x, y, z \in \{0, 1\}$, add the following tile types:

No carry in
No carry out



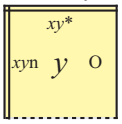
ii. If the carry bit being propagated stops at the second most significant bit, then for all $x \in \{0, 1\}$, add the following tile types:

Carry in
No carry out



(e) Rightmost tile type - for all $x, y \in \{0, 1\}$, add the following tile type:

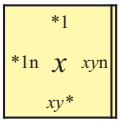
No carry in



(f) Second right *and* second leftmost tile type (when $w = 3$.)

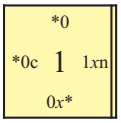
i. If there is no carry bit to propagate, then for all $x, y \in \{0, 1\}$, add the following tile types:

No carry in,
No carry out



ii. If the carry bit stops at this location, then for all $x \in \{0, 1\}$, add the following tile types:

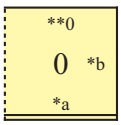
Carry in,
No carry out



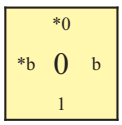
3. Tile types that perform special case increments.

(a) When the current row represents a power of 2 - add the following tile types:

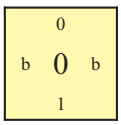
Leftmost



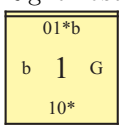
Second leftmost



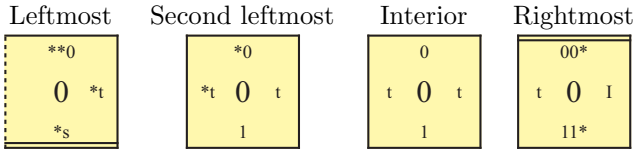
Interior



Rightmost



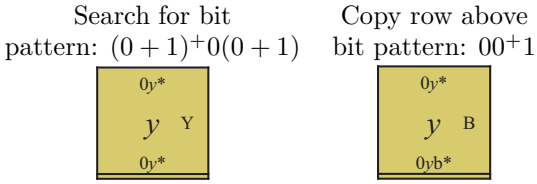
(b) When the current row represents 0 - add the following tile types:



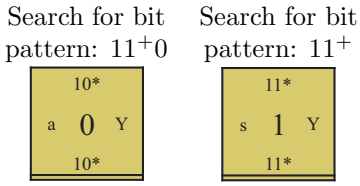
4. Tile types that copy the current value of the counter (right to left).

   (a) Rightmost tile type of copy row.

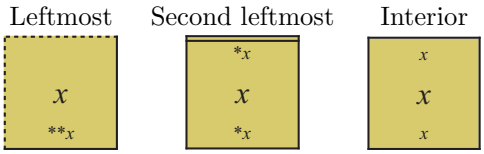
      i. Rightmost tile types that initiate a generic copy row - for all $y \in \{0, 1\}$, add the following tile types:



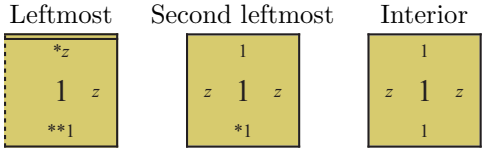
      ii. Rightmost tile types that search for a particular bit pattern - add the following tile types:



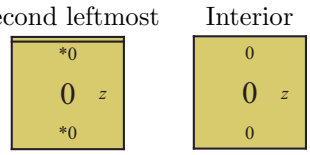
   (b) Copy row interior tile types - for all $x \in \{0, 1\}$, add the following tile types:



   (c) Copy row tile types that perform bit pattern search - for all $z \in \{\text{a}, \text{s}\}$ add the following tile types:



   (d) Tile types that terminate bit pattern search - for all $z \in \{\text{a}, \text{s}\}$ add the following tile types:



4.4 Overview Of The Planter Module

The next module is the planter module because it "plants the seeds" from which the ray modules will ultimately grow (the darkest gray squares in Figure 1). At the core of the planter module is a log-width, horizontal binary counter that counts every positive integer, starting at 1, in order. A key feature of the binary counter embedded in the planter module is that, after each integer is counted, a number of columns,

equal to the current value of the binary counter, plus the number of bits in the binary representation of this value, plus a few extra "dummy" spacing columns, self-assemble. This has the effect of spacing out successive ray modules according to the function f (given in the introduction).

4.5 Details Of Construction Of The Planter Module

The tile set for the planter consists of 94 tile types. We partition the tile types into four logical subgroups.

The first of the four subgroups is a standard binary counter that counts from 1 to infinity with the LSB of each integer having a y-coordinate of -1. The binary counter is based on an infinite fixed-width version of the binary counter in [10].

Suppose that $x \in \mathbb{Z}^+$ be the current value of the binary counter. The second subgroup receives x as input, and then in each subsequent row, subtracts one from the value of the number in the previous row until reaching 0, at which time a final "dummy" row consisting of all zeros self-assembles. This results in the self-assembly of exactly $x + 1$ rows following the binary counter row that represents x. The tile types that perform subtraction are based on the optimal binary counter (see [4]). Note that while subtraction is taking place, the current value of the binary counter is "remembered" via the rightmost bits in the east/west edge labels so that the value can be input to the third subgroup. Figure 5 shows a detailed example of the subtraction process for $n = 4$.

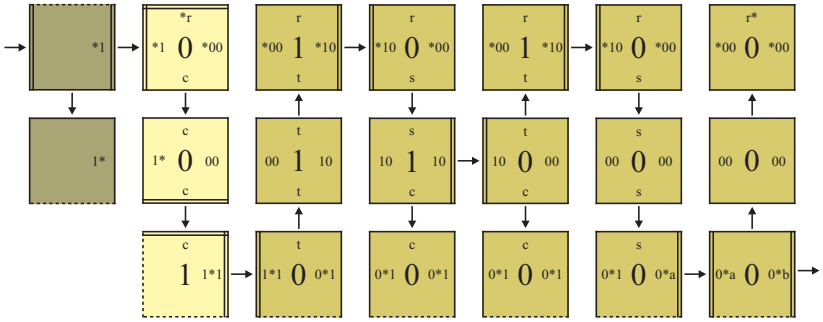


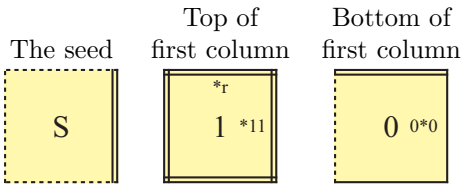
Figure 5: Subtraction in the planter. The first column of tiles represents the current value of the binary counter (4). The five rightmost columns of tiles perform subtraction.

Next, the third subgroup of tile types self-assembles into a square of size $\lfloor \lg x \rfloor + 1$ such that its north most edge labels are colored with the bits of x. The tile representing the LSB of x is the one that is placed in the upper right corner of the square. Notice that this has the affect of rotating and reflecting x up immediately below the x-axis. This is shown with detail in Figure 6. Notice that the rightmost tiles in Figure 5 bind with the leftmost tiles in Figure 6.

Finally, the fourth subgroup self-assembles three inert "dummy" spacing rows while allowing the current value of the binary counter to pass through. After these spacing rows attach, the next row of the binary counter self-assembles in which 1 is added to x. This process is repeated for all $x \in \mathbb{Z}^+$.

The following is the set of tile types that make up the planter module.

1. Seed tile types.



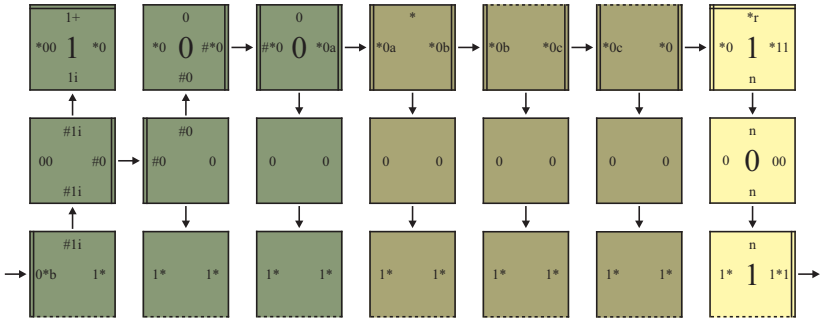
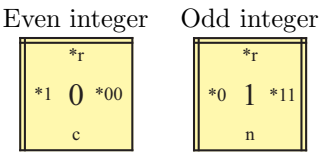


Figure 6: Rotation in the planter. The first three columns of tiles rotates and reflects the input (4) up immediately below the x-axis. The three subsequent columns of tiles are "dummy" spacing columns, and the final column of tiles represents the next value of the binary counter.

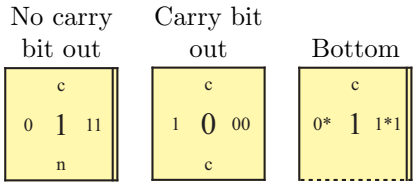
2. Binary counter tile types. The following tile types implement a fixed-width binary counter that counts every positive integer. Self-assembly proceeds naturally from LSB to MSB, which in our construction is top-to-bottom.

   (a) The first (topmost) tile type to attach (depending on whether the current row represents an even or odd integer) - add the following tile types:

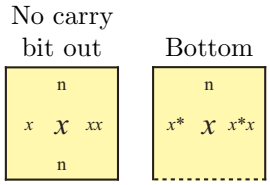
   

   (b) Interior tile types that perform increments; The 'c' and 'n' characters symbolize situations in which there is either a carry bit coming in from the top or there is no carry bit.

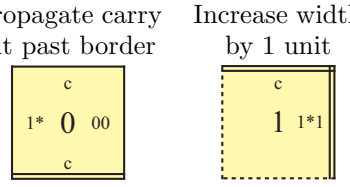
      i. If there is a carry bit coming in, add the following tile types:

      ii. If there is no carry bit coming in, then for all $x \in \{0, 1\}$, add the following tile types:

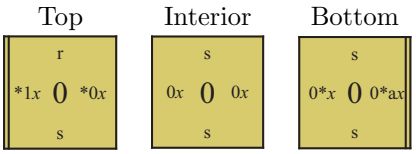
      

   (c) Tile types that increase the number of bits in the binary counter - add the following tile types:

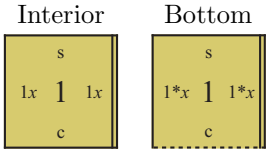
   

3. Subtraction tile types (a modification of the optimal binary counter given in [4]).

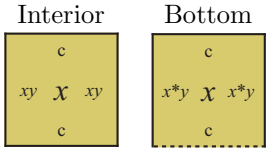
   (a) Tile types that search for the least significant 1 bit in the current column - for all $x \in \{0, 1\}$, add the following tile types:



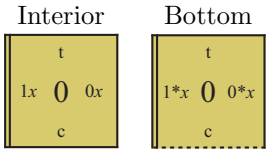
   (b) Tile types that find the least significant 1 bit in the current column - for all $x \in \{0, 1\}$, add the following tile types:



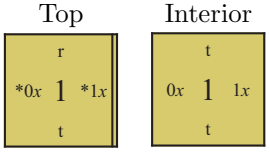
   (c) Tile types that copy the least significant bits to the right (bottom) of the least significant 1 bit in the current column - for all $x, y \in \{0, 1\}$, add the following tile types:



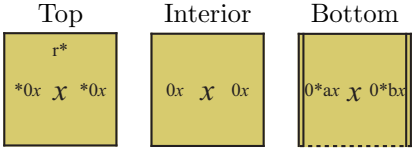
   (d) The first tile type to attach in a row that represents an odd integer. Note that all tile types that attach above this tile type in a given column will represent the bit 1 (this is the purpose of the 't' signal) - for all $x \in \{0, 1\}$, add the following tile types:



   (e) Tile types that convert all the least significant bits to the right(top) of the least significant 1 (in the previous column) to 1 - for all $x \in \{0, 1\}$, add the following tile types:



   (f) After subtraction reaches 0, one final column is added - for all $x \in \{0, 1\}$, add the following tile types:

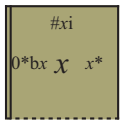


4. Rotation tile types.

(a) Tile types of the initial column.

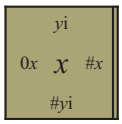
i. The first tile type - for all $x \in \{0, 1\}$, add the following tile types:

Bottom

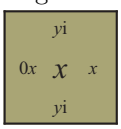


ii. Interior tile types - for all $x, y \in \{0, 1\}$, add the following tile types:

Second to attach

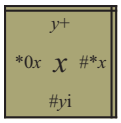


In general

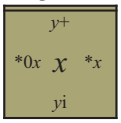


iii. Top most tile types - for all $x, y \in \{0, 1\}$, add the following tile types:

Special case



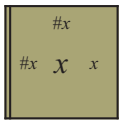
In general



(b) Tile types that shift '#' up and to the right (not the initial or final column) while simultaneously copying the current value of the binary counter (the fourth subgroup) left-to-right, and bottom-to-top.

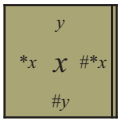
i. Shift right and then up - for all $x \in \{0, 1\}$, add the following tile types:

The first tile type in a column

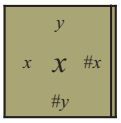


ii. Shift up and then right - for all $x, y \in \{0, 1\}$, add the following tile types:

Topmost

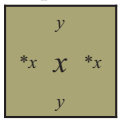


In general

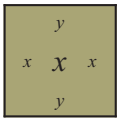


iii. Tile types that do not participate in shifting the '#' token but are above the main diagonal - for all $x, y \in \{0, 1\}$, add the following tile types:

Topmost

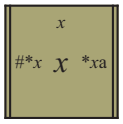


In general

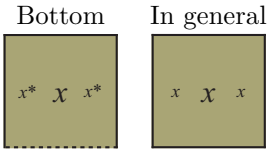


(c) The first tile type to attach in the final column - for all $x \in \{0, 1\}$, add the following tile types:

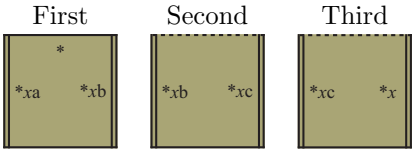
The first tile type in a column



(d) Tile types that do not participate in shifting the '#' token but are below the main diagonal - for all $x \in \{0, 1\}$, add the following tile types:



(e) Tile types that are responsible for the self-assembly of three inert spacing rows immediately after rotation - for all $x \in \{0, 1\}$, add the following tile types:



4.6 Overview Of The Computation Module

The final module is, for any TM M, an algorithmically generated tile set that, in conjunction with the ray and planter modules, achieves the simulation of M on (the binary representation of) every input $x \in \mathbb{Z}^+$. The simulation of M on $\text{bin}(x)$ proceeds vertically, immediately above the planter, while following the contour defined by the rightmost edge of the ray of width $x+2$ (Note that by our construction of the planter module, there is one, and only one ray of such width). As with other standard Turing machine constructions (see [15, 10, 12]), each row in our simulation represents a configuration of M. However, the frequency with which transitions occur is a novel feature of our construction, and is controlled by "color" signals that are received from the abutting ray module.

4.7 Details Of Construction Of The Computation Module

Our simulation of M on x proceeds vertically while following the contour defined by the rightmost edge of the ray of width $x + 2$. The right edge label of the rightmost tile type in every row of this ray sends a "color" signal that has the potential to initiate the self-assembly of a simulation row (if M halts on x, then eventually there will be no simulation rows to influence). The color signal defines the type of simulation row that subsequently self-assembles, whence self-assembly of every simulation row (except "blue" rows) proceeds left-to-right. In fact, all non-blue simulation rows, more or less, simply copy the previous configuration of M up one unit while, in some cases, also performing a shift-to-the-right by one unit.

Simulation begins with a row of tiles that represent the initial configuration of M with input x, which is represented by a row of $\lfloor \lg x \rfloor + 2$ red tile types (see below). The initial simulation row is adjacent to the rightmost tile of the initial row of the ray of width $x + 2$, and immediately above the x-axis. All subsequent simulation rows either copy the configuration of M up to the next row (yellow and indigo rows) or do so while also shifting the contents to the right by one unit (orange rows). However, if a green color signal is received, then the simulation row increases the size of the working tape by a single tile (to the right), and then performs a single computation step in the next (blue) row. Note that when the size of the tape increases, the ray of width $x+2$ does not shift to the right, but the ray of width $x+3$ does. This relationship between successive rays maintains the constant width gap of 2 tiles between the simulation of M on x and the left border of the ray of width $x + 3$.

If $M(x) \downarrow$, upward growth of the simulation halts, and a special signal is sent along a one-tile-wide path of tiles left-to-right along the top of the most recent simulation row. When the rightmost tile is encountered, the path continues down along the contour of the rightmost edge of the simulation until reaching the x-axis, at which point a black tile type is placed at the location $(f(x), 0)$. Since there is a constant gap (of width

2) between the right border of the simulation of M on x and the left border of the ray of width $x+3$, the halting signal proceeds unimpeded.

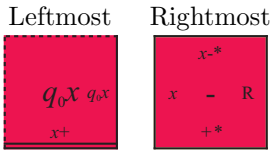
The following is the set of tiles that make up the simulation module.

Let $M = (Q, \Sigma, \Gamma, \text{-}, \delta, q_0, q_h)$ be a Turing machine where

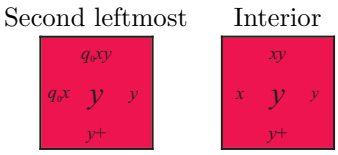
- Q is a finite set of states;
- $\Sigma = \{0, 1\}$ is the input alphabet;
- Γ is the tape alphabet;
- $\text{-} \in \Gamma - \Sigma$ is the blank symbol;
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$, is the transition function;
- $q_0 \in Q$ is the start state, and
- $q_h \in Q$ is the halting state.

We make the assumption that M is initially in q_0 reading the leftmost symbol of its input $n \in \mathbb{N}$, which is in the leftmost cell on a one-way infinite tape. Further, we stipulate that M never moves left when reading the leftmost symbol on its tape.

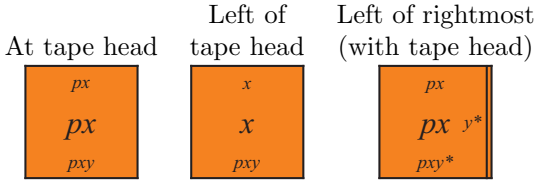
1. Red (seed) tile types.
   (a) For all $x \in \Gamma$, add the following tile types:



   (b) For all $x, y \in \Gamma$, add the following tile types:

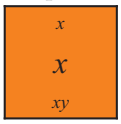


2. Orange (copy) tile types.
   (a) If the location contains, or is next to the tape head, then for all $x, y \in \Gamma$, and for all $p \in Q$, add the following tile types:

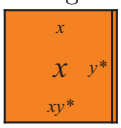


   (b) If the location is neither at or adjacent to the tape head, nor the rightmost location in the row, then for all $x, y \in \Gamma$, add the following tile types:

Copy
(no tape head)

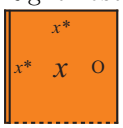


Left of rightmost



(c) Otherwise, for all $x \in \Gamma$, add the following tile types:

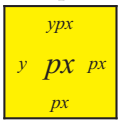
Rightmost



3. Yellow tile types (shift tape contents right one unit).

(a) For all $x, y \in \Gamma$, and for all $p \in Q$, add the following tile types:

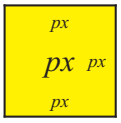
Interior
(with tape head)



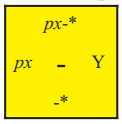
Right of
tape head

(b) For all $x \in \Gamma$, and for all $p \in Q$, add the following tile types:

Leftmost
(with tape head)

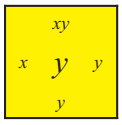


Rightmost and
adjacent to tape head



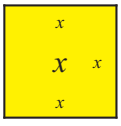
(c) For all $x, y \in \Gamma$, add the following tile types:

Interior

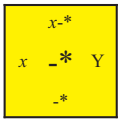


(d) For all $x \in \Gamma$, add the following tile types:

Leftmost
(no tape head)



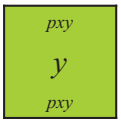
Rightmost



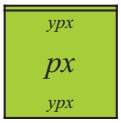
4. Green tile types (pre-transition):

(a) For all $x, y \in \Gamma$, and for all $p \in Q$, add the following tile types:

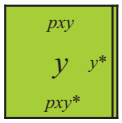
Interior (left of tape head)



Interior (with tape head)

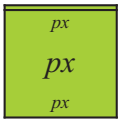


Left of rightmost (right of tape head)



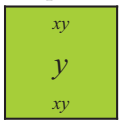
(b) For all $x \in \Gamma$, and for all $p \in Q$, add the following tile types:

Leftmost (with tape head)

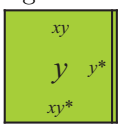


(c) For all $x, y \in \Gamma$, add the following tile types:

Interior (no tape head)

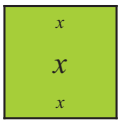


Left of rightmost

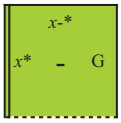


(d) For all $x \in \Gamma$, add the following tile types:

Leftmost (no tape head)



Rightmost

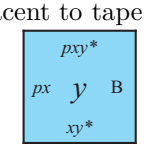


5. Blue tile types (Turing machine transition).

(a) Tile types that do not perform a transition.

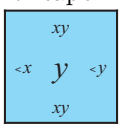
i. For all $x, y \in \Gamma$, and for all $p \in Q$, add the following tile types:

Rightmost and adjacent to tape head

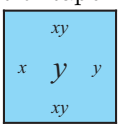


ii. For all $x, y \in \Gamma$, add the following tile types:

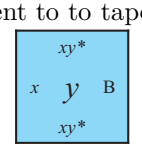
Interior (left of tape head)



Interior (right of tape head)

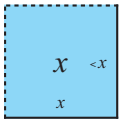


Rightmost and not adjacent to to tape head



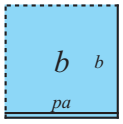
iii. For all $x \in \Gamma$, add the following tile types:

Leftmost



(b) Tile types that perform a right transition. If $\exists a, b \in \Gamma$ such that $(q, b, r) = \delta(p, a)$, then add the following tile types:

Interior and right of tape head



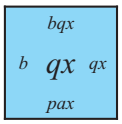
i. For all $x, y \in \Gamma$, add the following tile types:

Interior and right of tape head

ii. For all $x \in \Gamma$, add the following tile types:

Start of transition at interior

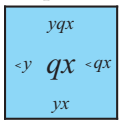
Right of tape head after transition



(c) Tile types that perform a left transition. If $\exists a, b \in \Gamma$ such that $(q, b, L) = \delta(p, a)$, then add the following tile types:

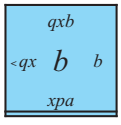
i. For all $x, y \in \Gamma$, add the following tile types:

Interior and left of tape head

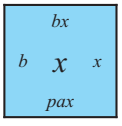


ii. For all $x \in \Gamma$, add the following tile types:

Start of transition at interior



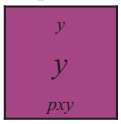
Right of transition



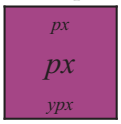
6. Indigo (copy tape contents straight up) tile types.

(a) For all $x, y \in \Gamma$, and for all $p \in Q$, add the following tile types:

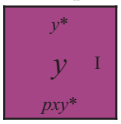
Interior (left of tape head)



Rightmost, and next to tape head

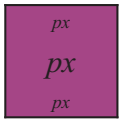


Rightmost, and not next to tape head



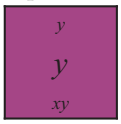
(b) For all $x \in \Gamma$, and for all $p \in Q$, add the following tile types:

Leftmost (with tape head)

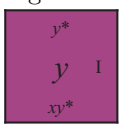


(c) For all $x, y \in \Gamma$, add the following tile types:

Interior without tape head

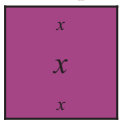


Rightmost



(d) For all $x \in \Gamma$, add the following tile types:

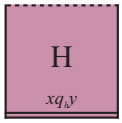
Leftmost (without tape head)



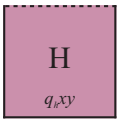
7. Violet (halting) tile types.

(a) For all $x, y \in \Gamma$, add the following tile types:

Initial interior halting tile type



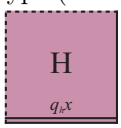
Right of initial halting tile type



Fill in remainder of final row

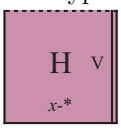
(b) For all $x \in \Gamma$, add the following tile types:

Initial halting tile type (leftmost)



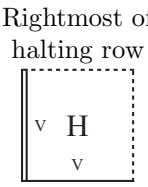
Leftmost tile type (no tape head)

Second rightmost tile type

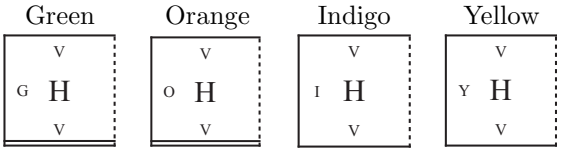


8. Tile types that snake down the rightmost edge of a halting simulation.

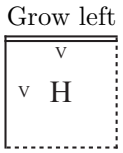
(a) The first tile type to attach:



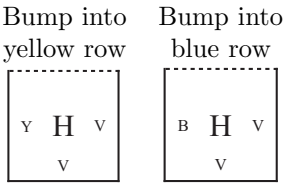
(b) Grow down:



(c) Initiate left growth:



(d) Grow left one unit:



9. If M halts on input n then the following tile type (the only "black" tile type in our construction) is placed at the point $(f(n), 0)$:

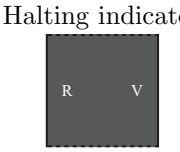


Figure 7 shows a trivial example of the simulation of a particular Turing machine M on input 01 (the binary representation of 1, with a leading 0).

4.8 Proof of Correctness

Let $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ be as in section 1, stipulating that $f(0) = -1$. For each $n \in \mathbb{N}$, define the rectangle

$$Q_n = \{f(n-1) + 2, \ldots, f(n) - 1\} \times \{-2, -1\},$$

and define the following assemblies.

(i) σ_n is the portion of the planter lying in Q_n.

(ii) ρ_n is the ray of width $n + 2$, translated so that its base is the leftmost $2(n + 2)$ tiles in σ_n.

(iii) $\sigma_n^* = \sigma_n + \rho_n$, where "+" is the operation defined in section 3.

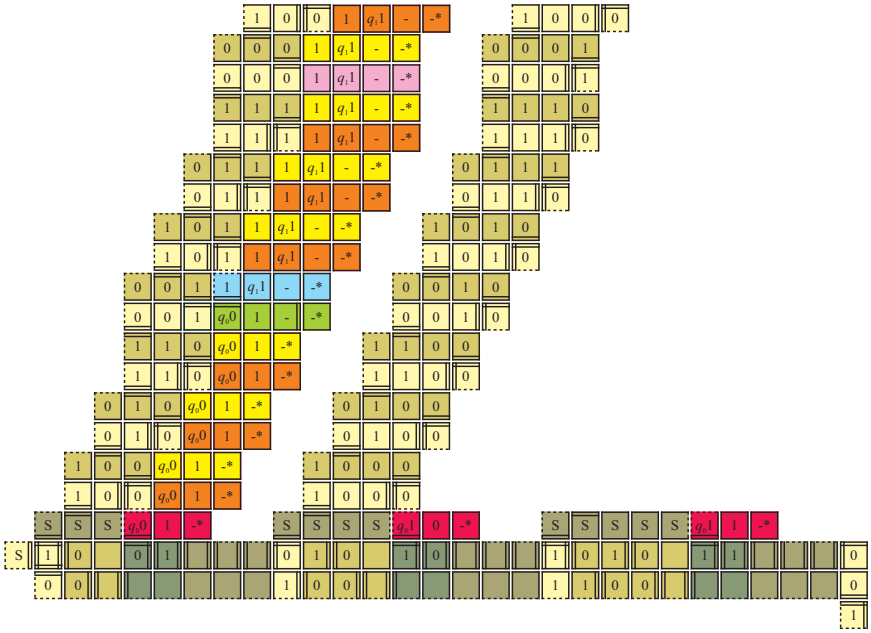


Figure 7: A closer look at the simulation of M on input 1 (influenced by a ray of width 3). All three modules can be seen in this figure.

(iv) γ_n is the assembly that simulates $M(n)$ as in section 4.4.

(v) σ^* is our planter.

(vi) $\alpha_n = \sigma_n^* + \gamma_n$.

(vii) $\alpha = \sigma^* + \sum_{n=0}^{\infty} \alpha_n$.

Let

$$\mathcal{T}_M = (T_M, \sigma, \tau)$$

be our tile assembly system, noting that σ consists of a single tile at the origin. Define the following subsets of $\mathcal{T}_M$.

(i) T_R is the set of tile types used in the ray module, together with the benign tile type, all of whose edges are given a binding strength of 1 with no color, appearing in the rectangles σ_n.

(ii) T_C is the set of tile types in T_R, together with those occurring in the computation module.

(iii) T_P is the set of tile types in the planter module.

For each $n \in \mathbb{Z}^+$, define the tile assembly systems

$$\mathcal{T}_{R,n} = (T_R, \sigma_n, \tau),$$
$$\mathcal{T}_{C,n} = (T_C, \sigma_n, \tau),$$
$$\mathcal{T}_{C,n}^* = (T_C, \sigma_n^*, \tau).$$

Lemma 4.2. For each $n \in \mathbb{Z}^+$, $\mathcal{A}_{\square}[\mathcal{T}_{R,n}] = \{\sigma_n^*\}$.

Proof. Define the infinite, τ-T_R-assembly sequence, $\vec{\alpha}$, to be that which is implicit in Figures 2, 3, and 4. It is easy to see from our construction that every tile that binds in $\vec{\alpha}$ does so uniquely. Furthermore, every such tile addition occurs with exactly strength 2. It is clear that $\text{res}(\vec{\alpha})$ is terminal, whence $\vec{\alpha}$ is a locally deterministic assembly sequence. □

Lemma 4.3. For each $n \in \mathbb{Z}^+$, σ_n^* is a pseudoseed of $\mathcal{T}_{C,n}$.

Proof. First, note that $\sigma_n^* \in \mathcal{A}[\mathcal{T}_{C,n}]$ because $\sigma_n^* \in \mathcal{A}[\mathcal{T}_{R,n}]$ and $T_R \subset T_C$.

Let $\alpha \in \mathcal{A}[\mathcal{T}_{C,n}]$. Let $\vec{\alpha}'$ be the τ-T_C-assembly sequence such that $\text{res}(\vec{\alpha}') = \alpha$. It is clear, since $\vec{\alpha}$ is locally deterministic and because of the use of module indicators in our construction, that when $\vec{\alpha}'$ assigns a tile type to a location in ρ_n, it does so in agreement with $\vec{\alpha}$ from Lemma 4. Thus, we can use $\vec{\alpha}$ to "extend" $\text{res}(\vec{\alpha}')$ and thereby fill in the remainder of ρ_n. This gives us an assembly $\alpha' \in \mathcal{A}\left[\mathcal{T}_{C,n}^*\right]$ such that $\alpha \sqsubseteq \alpha'$. □

Lemma 4.4. For each $n \in \mathbb{Z}^+$, $\mathcal{A}_\square\left[\mathcal{T}_{C,n}^*\right] = \{\alpha_n\}$.

Proof. Define the τ-$T_{C,n}$-assembly sequence $\vec{\alpha}_n$ as that which self-assembles each simulation row one at a time, and according to the following rule: if the current row is not "blue," then, by our construction, self-assembly must proceed left to right; however, if the current row is "blue," then $\vec{\alpha}_n$ first attaches the initial tile, via a single τ-strength bond along its south edge, and then the remaining tiles in order of increasing distance from the origin. It is clear from our construction that every tile that binds in $\vec{\alpha}_n$ does so uniquely, and with exactly strength τ. Since $\text{res}(\vec{\alpha}_n)$ is terminal, it follows that $\vec{\alpha}_n$ is locally deterministic. □

Lemma 4.5. For each $n \in \mathbb{Z}^+$, $\mathcal{A}_\square[\mathcal{T}_{C,n}] = \{\alpha_n\}$.

Proof. This follows immediately from Lemma 4, Lemma 5, and the pseudoseed lemma. □

Define the tile assembly system

$$\mathcal{T}_P = (T_P, \sigma, \tau).$$

Lemma 4.6. $\mathcal{A}_\square[\mathcal{T}_P] = \{\sigma^*\}$.

Proof. Define the infinite τ-T_P-assembly sequence, $\vec{\alpha}$, to be that which self-assembles σ^* one row at a time, and implicit from Figures 5 and 6. Note that $\vec{\alpha}$ starts from the seed tile located at the origin. It is clear from our construction that every tile that binds via $\vec{\alpha}$ does so uniquely. Furthermore, every such tile addition occurs with exactly strength 2. It is clear that $\text{res}(\vec{\alpha})$ is terminal, whence $\vec{\alpha}$ is a locally deterministic assembly sequence. □

Lemma 4.7. σ^* is a pseudoseed of $\mathcal{T}_M$.

Proof. First, note that $\sigma^* \in \mathcal{A}[\mathcal{T}_M]$ because $\sigma^* \in \mathcal{A}[\mathcal{T}_P]$ and $T_P \subset T_M$.

Let $\alpha \in \mathcal{A}[\mathcal{T}_M]$. Let $\vec{\alpha}'$ be the τ-T_M-assembly sequence such that $\text{res}(\vec{\alpha}') = \alpha$. It is clear, since $\vec{\alpha}$ is locally deterministic and because of the use of module indicators in our construction, that when $\vec{\alpha}'$ assigns a tile to a location that is in σ^*, it does so in agreement with $\vec{\alpha}$ from Lemma 8. Thus, we can use $\vec{\alpha}$ to "extend" $\text{res}(\vec{\alpha}')$ and thereby fill in the remainder of σ^*. This gives us an assembly $\alpha' \in \mathcal{A}\left[\mathcal{T}_{C,n}^*\right]$ such that $\alpha \sqsubseteq \alpha'$. □

Lemma 4.8. σ^* is a $\vec{\sigma}$-T_C-T_M-multiseed.

Proof. Let $\vec{\sigma} = (\sigma_n \,|n \in \mathbb{Z}^+)$. By our construction of the planter, it is clear that part (1), of definition 3, is satisfied. To see that (2) is satisfied, we appeal to: Lemma 7, our construction of the ray, and the fact that the planter spaces out each σ_n sufficiently. Moreover, it is clear that (5) holds because of our use of module indicator symbols in our construction. We now turn our attention to parts (3) and (4).

Let $i \in \mathbb{Z}^+$, and $\alpha \in \mathcal{A}[\mathcal{T}_i]$. Let $\vec{\alpha}'$ be an assembly sequence such that $\alpha = \text{res}(\vec{\alpha}')$. Since (1) holds, we can define the assembly sequence $\vec{\alpha} = (\sigma^*, \ldots, \alpha)$, and let $\alpha^* = \text{res}(\vec{\alpha})$. It is clear that $\alpha^* \in \mathcal{A}[\mathcal{T}_M^*]$, and $\alpha \sqsubseteq \alpha^*$, whence (3) holds.

Finally, let $\alpha^* \in \mathcal{A}[\mathcal{T}_M]$. By (1), (2), Lemma 7, and our construction of the planter, it is easy to see that (4) holds.

□

Lemma 4.9. $\mathcal{A}_\square[\mathcal{T}_M] = \{\alpha\}$.

Proof. Let $\mathcal{T}_M^* = (T_M, \sigma^*, \tau)$. By Lemma 6, 7, and 9, and the multiseed lemma, $\mathcal{A}_\square[\mathcal{T}_M^*] = \{\alpha\}$. It follows by Lemma 8 and the pseudoseed lemma that $\mathcal{A}_\square[\mathcal{T}_M] = \{\alpha\}$. □

5 A Decidable Set That Does Not Self-Assemble

We now show that there are decidable sets $D \subseteq \mathbb{Z}^2$ that do not self-assemble in the Tile Assembly Model.

For each $r \in \mathbb{N}$, let

$$D_r = \{ (m,n) \in \mathbb{Z}^2 \mid |m| + |n| = r \}.$$

This set is a "diamond" in $\mathbb{Z}^2$ (i.e., a circle in the taxicab metric) with radius r and center at the origin. For each $A \subseteq \mathbb{N}$, let

$$D_A = \bigcup_{r \in A} D_r.$$

As illustrated in Figure 8, this set is the "system of concentric diamonds" centered at the origin with radii in A.

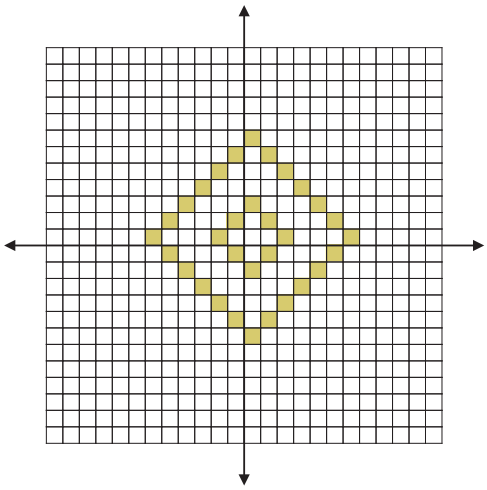


Figure 8: The set $D_{\{2,6\}}$

The following trivial observation is used in the proof of Lemma 5.2.

Observation 5.1. $|D_{<r}| = \left|\{ (m,n) \in \mathbb{Z}^2 \mid |m| + |n| < r \}\right| = 2(r-1)^2 + 2r - 1$.

Proof.

$$\begin{aligned} |D_{<r}| &= \left|\{ (m,n) \in \mathbb{Z}^2 \mid |m| + |n| < r \}\right| \\ &= 2r - 1 + 2 \cdot \sum_{i=0}^{r-2} (2i+1) \\ &= 2r - 1 + 2(r-1)^2. \end{aligned}$$

□

Lemma 5.2. If $A \subseteq \mathbb{N}$ and D_A self-assembles, then $A \in \text{DTIME}\left(2^{4n}\right)$.

The proof of this lemma exploits the fact that a tile assembly system in which D_A self-assembles must, for sufficiently large r, decide the condition $r \in A$ from *inside* the diamond D_r.

Proof. If $|A| < \infty$ then we are done, so assume otherwise (i.e., A is infinite).

Assume the hypothesis. Then there exists a 2-TAS $\mathcal{T} = (T, \sigma, \tau)$ in which the set D_A self-assembles. Fix some enumeration $\vec{a}_1, \vec{a}_2, \vec{a}_3 \ldots$ of $\mathbb{Z}^2$, and let M be the TM, defined as follows, that simulates the self-assembly of $\mathcal{T}$.

Require: $r \in \mathbb{N}$
  $\alpha := \sigma$
  while $D_r \cap \text{dom } \alpha = \emptyset$ **do**
    choose the least $j \in \mathbb{N}$ such that $\vec{a}_j \in \partial\alpha_i$
    choose $t \in T$ such that $\vec{a}_j \in \partial_t\alpha_i$
    $\alpha := \alpha + (\vec{a}_j \mapsto t)$
  end while
  if $\alpha(\vec{a}_j) \in B$ **then**
    accept
  else
    reject
  end if

Observe that the *while* loop will terminate after at most $|D_{<r}|+1-|\text{dom } \alpha|$ iterations, with $D_r \cap \text{dom } \alpha \neq \emptyset$ whence M halts on all inputs. If $r \in A$, then the weak self-assembly of D_A tells us that for every $\alpha \in \mathcal{A}_\square[\mathcal{T}]$, $\alpha(D_r) \subseteq B$. Since we have $\vec{a}_j \in D_r$ when the *while* loop terminates, M accepts, and $r \in L(M)$. If $r \in L(M)$, then $\alpha(\vec{a}_j) \in B$, whence $r \in A$.

By Observation 5.1, the *while* loop performs at most $2(r-1)^2 + 2r - 1$ iterations, and in each iteration, we are doing at most $O\left(r^2\right)$ amount of additional work checking $D_r \cap \text{dom } \alpha = \emptyset$, and maintaining $\partial^\tau\alpha_i$. Thus, M decides A in $O\left(2^{4n}\right)$ computation steps, where $n = \lfloor \log r \rfloor + 1$ (i.e., the length of r).

Note: Implicit in this proof is the fact, proven by Irani, Naor, and Rubinfeld [6], that write-once memory is equivalent to time. □

We now have the following result.

Theorem 5.3 (second main theorem)**.** There is a decidable set $D \subseteq \mathbb{Z}^2$ that does not self-assemble.

Proof. By the time hierarchy theorem [5], there is a set $A \subseteq \mathbb{N}$ such that

$$A \in \text{DTIME}\left(2^{5n}\right) - \text{DTIME}\left(2^{4n}\right).$$

Let $D = D_A$. Then D is decidable and, by Lemma 5.2, D does not self-assemble. □

Note that both Lemma 5.2 and (hence) Theorem 5.3 hold for *any* value of $\tau > 0$.

6 Conclusion

Our first main theorem says that, for every computably enumerable set $A \subseteq \mathbb{Z}^+$, the representation $X_A = \{(f(n), 0) | n \in A\}$ self-assembles. This representation of A is somewhat sparse along the x-axis, because our f grows quadratically. A linear function f would give a more compact representation of A. We conjecture that our first main theorem does *not* hold for any linear function, but we do not know how to prove this.

Let D be the set presented in the proof of our second main theorem. It is easy to see that the condition $(m, n) \in D$ is decidable in time polynomial in $|m| + |n|$, but $|m| + |n|$ is exponential in the length of the

binary representation of (m, n), so this only tells us that $D \in \mathrm{E} = \mathrm{DTIME}\left(2^{\mathrm{linear}}\right)$. Is there a set $D \subseteq \mathbb{Z}^2$ such that $D \in \mathrm{P}$, and D does not self-assemble?

More generally, we hope that our results lead to further research illuminating the interplay between geometry and computation in self-assembly.

Acknowledgment

The authors wish to thank David Doty and Aaron Sterling for useful discussions. The authors would also like to thank an anonymous referee for several useful comments.

References

[1] Leonard Adleman, *Towards a mathematical theory of self-assembly*, Tech. report, University of Southern California, 2000.

[2] Jonathan Bachrach and Jacob Beal, *Building spatial computers*, Tech. report, MIT CSAIL, 2007.

[3] Jacob Beal and Gerald Sussman, *Biologically-inspired robust spatial programming*, Tech. report, MIT, 2005.

[4] Qi Cheng, Ashish Goel, and Pablo Moisset de Espanés, *Optimal self-assembly of counters at temperature two*, Proceedings of the First Conference on Foundations of Nanoscience: Self-assembled Architectures and Devices, 2004.

[5] J. Hartmanis and R. E. Stearns, *On the computational complexity of algorithms*, Transactions of the American Mathematical Society **117** (1965), 285–306.

[6] S. Irani, M. Naor, and R. Rubinfeld, *On the time and space complexity of computation using write-once memories, or Is pen really much worse than pencil?*, Theory of Computing Systems **25** (1992), 141–159.

[7] James I. Lathrop, Jack H. Lutz, and Scott M. Summers, *Strict self-assembly of discrete Sierpinski triangles*, Theoretical Computer Science **410** (2009), 384–405.

[8] John H. Reif, *Molecular assembly and computation: From theory to experimental demonstrations*, Proceedings of the Twenty-Ninth International Colloquium on Automata, Languages and Programming, 2002, pp. 1–21.

[9] Paul W. K. Rothemund, *Theory and experiments in algorithmic self-assembly*, Ph.D. thesis, University of Southern California, December 2001.

[10] Paul W. K. Rothemund and Erik Winfree, *The program-size complexity of self-assembled squares (extended abstract)*, STOC '00: Proceedings of the thirty-second annual ACM Symposium on Theory of Computing (New York, NY, USA), ACM, 2000, pp. 459–468.

[11] Nadrian C. Seeman, *Nucleic-acid junctions and lattices*, Journal of Theoretical Biology **99** (1982), 237–247.

[12] David Soloveichik and Erik Winfree, *Complexity of self-assembled shapes*, SIAM Journal on Computing **36** (2007), no. 6, 1544–1569.

[13] Hao Wang, *Proving theorems by pattern recognition – II*, The Bell System Technical Journal **XL** (1961), no. 1, 1–41.

[14] ———, *Dominoes and the AEA case of the decision problem*, Proceedings of the Symposium on Mathematical Theory of Automata (New York, 1962), Polytechnic Press of Polytechnic Inst. of Brooklyn, Brooklyn, N.Y., 1963, pp. 23–55.

[15] Erik Winfree, *Algorithmic self-assembly of DNA*, Ph.D. thesis, California Institute of Technology, June 1998.