Self-Assembly of Decidable Sets[*]

Matthew J. Patitz[†] and Scott M. Summers.[‡]

Abstract

The theme of this paper is computation in Winfree's Abstract Tile Assembly Model (TAM). We first review a simple, well-known tile assembly system (the "wedge construction") that is capable of universal computation. We then extend the wedge construction to prove the following result: if a set of natural numbers is decidable, then it and its complement's canonical two-dimensional representation self-assemble. This leads to a novel characterization of decidable sets of natural numbers in terms of self-assembly. Finally, we show that our characterization is robust with respect to various (restrictive) geometrical constraints.

1 Introduction

In his 1998 Ph.D. thesis, Erik Winfree [10] introduced the (abstract) Tile Assembly Model (TAM) - a mathematical model of laboratory-based nanoscale self-assembly. The TAM is also an extension of Wang tiling [8, 9]. In the TAM, molecules are represented by un-rotatable, but translatable two-dimensional square "tiles," each side of which having a particular glue "color" and "strength" associated with it. Two tiles that are placed next to each other *interact* if the glue colors on their abutting sides match, and they *bind* if the strength on their abutting sides matches, and is at least a certain "temperature." Extensive refinements of the TAM were given by Rothemund and Winfree in [6, 5], and Lathrop, Lutz and Summers [3] gave a treatment of the model that does not discriminate against the self-assembly of infinite structures.

In this paper, we explore the notion of *computation* in the TAM - what is it, and how is it accomplished? Despite its deliberate over-simplification, the TAM is a computationally expressive model. For instance, Winfree proved [10] that in two or more spatial dimensions, the TAM is capable of Turing-universal computation. In other words, it is possible to construct, for any Turing machine M and any input string w, a finite assembly system (i.e., finite set of tile types) that tiles the first quadrant and encodes the set of all configurations that M goes through when processing the input string w. This implies that the process of self-assembly can (1) be directed algorithmically, and (2) be used to evaluate computable functions.

One can also regard the process of self-assembly itself as computation that takes as input some initial configuration of tiles and produces output in the form of some particular connected shape, and *nothing* else (i.e., *strict* self-assembly [3]). The self-assembly of shapes, and their associated Kolmogorov (shape) complexity, was studied extensively by Soloveichik and Winfree in [7], where they proved the counter-intuitive fact that sometimes fewer tile types are required to self-assemble a "scaled-up" version of a particular shape than the actual un-scaled shape.

Another flavor of computation in the TAM is the self-assembly of a computationally interesting set (or pattern) on top of a much larger possibly "less interesting" set that is used for auxiliary computations (so-called *weak* self-assembly). We say that a set of points A weakly self-assembles if there is a finite tile assembly

[*] This research was supported in part by National Science Foundation Grants 0652569 and 0728806. A preliminary version of this research was presented at the Sixth International Conference on Unconventional Computation, August 25-28 2008, Vienna, Austria.

[†] Department of Computer Science, Iowa State University, Ames, IA 50011, USA. mpatitz@cs.iastate.edu.

[‡] Department of Computer Science, Iowa State University, Ames, IA 50011, USA. 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

system that places “black” tiles on, and only on, the points that are in A. One can also view weak self-assembly as a the process of a tile system “painting” a picture of the set A onto a much larger canvas of tiles. It is clear that if A weakly self-assembles, then A is necessarily computably enumerable. Moreover, Lathrop, Lutz, Patitz and Summers [2] discovered that the converse of the previous statement holds in the following sense. If the set A is computably enumerable, then a “simple” two-dimensional representation of A, as points along the x-axis, weakly self-assembles. This result is interesting because its proof requires the simulation of a particular Turing machine M on *infinitely* many inputs (i.e., for all $x \in \mathbb{N}$) in the two-dimensional discrete Euclidean plane $\mathbb{Z}^2$.

In this paper, we continue the work of Lathrop, Lutz, Patitz and Summers [2]. Specifically, we focus our attention on the weak self-assembly of canonical two-dimensional representations of decidable sets in the TAM. We first reproduce Winfree’s proof of the universality of the TAM [10] in the form of a simple construction called the “wedge construction.” The wedge construction self-assembles the *computation history* of an arbitrary TM M on input w in the space to the right of the y-axis, above the x-axis, and above the line $y = x - |w| - 2$. We then prove our first main result, which follows from a straight-forward extension of the wedge construction and gives a new characterization of decidable languages of natural numbers in terms of (weak) self-assembly. That is, we prove that a set $A \subseteq \mathbb{N}$ is decidable if and only if $A \times \{0\}$ and $A^c \times \{0\}$ weakly self-assemble. Technically speaking, our characterization is (exactly) the first main theorem from Lathrop, Lutz, Patitz and Summers [2] with “computably enumerable” replaced by “decidable,” and $f(n) = n$. Finally, we show that our construction is robust with respect to an infinite class of restrictive geometrical constraints. In other words, our characterization can be carried out in the set of integer lattice points lying *above* the x-axis yet *below* the line $y = \frac{1}{a} \cdot x$, for any $a \in \mathbb{N}$.

2 The Tile Assembly Model

We now give a brief intuitive sketch of the abstract TAM. See [10, 6, 5, 3] for other developments of the model. We work in the 2-dimensional discrete Euclidean space. We write $U_2 = \{(0,1), (1,0), (0,-1), (-1,0)\}$.

Intuitively, a tile type t is a unit square that can be translated, but not rotated, having a well-defined “side $\vec{u}$” for each $\vec{u} \in U_2$. Each side $\vec{u}$ of t has a “glue” of “color” $\text{col}_t(\vec{u})$ - a string over some fixed alphabet Σ - and “strength” $\text{str}_t(\vec{u})$ - a natural number - specified by its type t. Two tiles t and t' that are placed at the points $\vec{a}$ and $\vec{a} + \vec{u}$ respectively, *bind* with *strength* $\text{str}_t(\vec{u})$ if and only if $(\text{col}_t(\vec{u}), \text{str}_t(\vec{u})) = (\text{col}_{t'}(-\vec{u}), \text{str}_{t'}(-\vec{u}))$.

Given a set T of tile types, an *assembly* is a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$. An assembly is *$\tau$-stable* if it cannot be broken up into smaller assemblies without breaking bonds of total strength at least τ, for some $\tau \in \mathbb{N}$.

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 deal exclusively with tile assembly systems in which $\tau = 2$. 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 $\alpha \in \mathcal{A}_\Box[\mathcal{T}]$, 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 $\mathcal{A}[\mathcal{T}]$ is partially ordered by the relation $\longrightarrow$ defined by

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

We say that $\mathcal{T}$ is *directed* if the relation $\longrightarrow$ is directed, i.e., if for all $\alpha, \alpha' \in \mathcal{A}[\mathcal{T}]$, there exists $\alpha'' \in \mathcal{A}[\mathcal{T}]$ 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 $\alpha \in \mathcal{A}[\mathcal{T}]$ 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 [7] 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. Intuitively, an assembly sequence $\vec{\alpha}$ is locally deterministic if (1) each tile added in $\vec{\alpha}$ "just barely" binds to the existing assembly; (2) if a tile of type t_0 at a location $\vec{m}$ and its immediate "output-neighbors" are deleted from the *result* of $\vec{\alpha}$, then no tile of type $t \neq t_0$ can attach itself to the thus-obtained configuration at location $\vec{m}$; and (3) the result of $\vec{\alpha}$ is terminal.

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

Throughout this paper, tiles are depicted as squares whose various sides are dotted lines, solid lines, or doubled lines, indicating whether the glue strengths on these sides are 0, 1, or 2, respectively. Thus, for example, a tile of the type shown in Figure 3 has glue of strength 0 on the left and bottom, glue of color 'a' and strength 2 on the top, and glue of color 'b' and strength 1 on the right. This tile also has a label 'L', which plays no formal role but may aid our understanding and discussion of the construction.

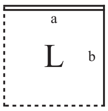


Figure 1: An example tile type.

3 The Wedge Construction

In this section, we review the "wedge construction" - a simple, well-known technique used to carry out the simulation of an arbitrary Turing machine on some binary string in the first quadrant of the discrete Euclidean plane. We will later extend the wedge construction to prove a new characterization of decidable languages.

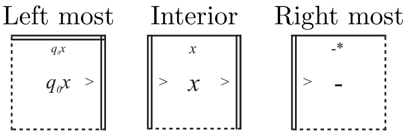
Lemma 3.1 (The wedge construction)**.** For every single-tape Turing machine M and input $w \in \{0,1\}^*$, there exists a tile assembly system $\mathcal{T}_{M(w)}$, which simulates M on w in the following way.

1. $\mathcal{T}_{M(w)}$ simulates the computation of $M(w)$, with the configuration of $M(x)$ after n steps represented by the line $y = n$ in the terminal assembly of $\mathcal{T}_{M(w)}$,
2. if M halts on w after k steps, then the line $y = k + 1$ in the terminal assembly of $\mathcal{T}_{M(w)}$ contains one and only one "halting" tile that binds via a single strength-2 bond on its south edge, and
3. $\mathcal{T}_{M(w)}$ is locally deterministic, and therefore directed.

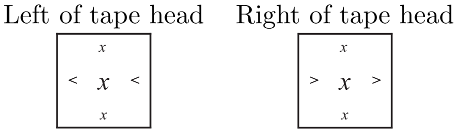
Proof. Our proof is by construction. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ be a Turing machine and $w \in \{0,1\}^*$. Assume, without loss of generality, that M is a Turing machine having a one-way infinite-to-the-right tape such that the tape head of M never attempts to move left while reading the left most tape cell. We define the finite set of tile types $T_{M(w)}$ as follows.

Definition of $T_{M(w)}$:

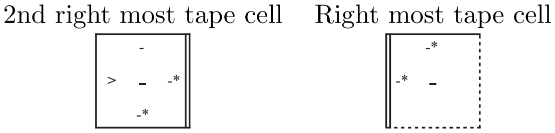
1. For all $x \in \Gamma$, add the seed row tile types:



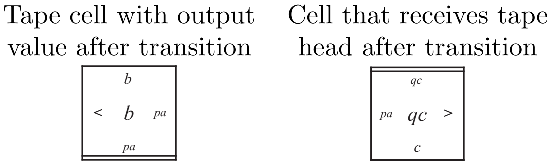
2. For all $x \in \Gamma$, add the tile types:



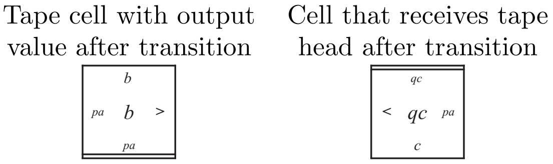
3. Add the following two tile types that grow the tape to the right:



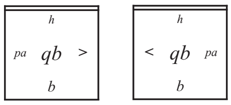
4. For all $p, q \in Q$, and all $a, b, c \in \Gamma$ satisfying $(q, b, \mathrm{R}) = \delta(p, a)$ and $q \notin \{q_{\text{accept}}, q_{\text{reject}}\}$ (i.e. for each transition moving the tape head to the right into a non-accepting state), add the tile types:


Tape cell with output value after transition
Cell that receives tape head after transition
b
< b pa
pa
qc
pa qc >
c

5. For all $p, q \in Q$, and all $a, b, c \in \Gamma$ satisfying $(q, b, \mathrm{L}) = \delta(p, a)$ and $q \notin \{q_{\text{accept}}, q_{\text{reject}}\}$ (i.e. for each transition moving the tape head to the left into a non-accepting state), add the tile types:


Tape cell with output value after transition
Cell that receives tape head after transition
b
pa b >
pa
qc
< qc pa
c

6. For all $p \in Q$, $a, b \in \Gamma$, and all $h \in \{\text{reject}, \text{accept}\}$ satisfying $\delta(q, b) \in \{q_{\text{accept}}, q_{\text{reject}}\} \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ (i.e. for each transition moving the tape head into a halting state), add the tile types:



Definition of σ_w: We now define the finite seed assembly σ_w of $\mathcal{T}_{M(w)}$. Let s_{left}, s_{interior} and s_{right} be the "Left most," "Interior," and "Right most" tile types, respectively, defined above in the first group of "seed row" tile types. Define σ_w as follows. $\sigma_w(0, 0) = s_{\text{left}}$, where each occurrence of x in s_{left} is replaced by $w[0]$ (i.e., the first bit of w); for all $0 < i < |w|$, $\sigma_w(i, 0) = s_{\text{interior}}$, with each occurrence of x in s_{interior} replaced by $w[i]$; let $\sigma_w(0, |w|) = s_{\text{right}}$; finally, let σ_w be undefined at all other points in $\mathbb{Z}^2$.

It is routine to verify that $\mathcal{T}_{M(w)}$ satisfies property (1) of the conclusion of the lemma. We now argue that $\mathcal{T}_{M(w)}$ is locally deterministic. To do so, we first define an assembly sequence $\vec{\alpha}$, leading to a terminal assembly $\alpha = \text{res}(\vec{\alpha})$, in which (1) the j^{th} configuration C_j of M is encoded in the row $R_j = (\{0, \ldots, |w| - 1 + j\} \times \{j\})$, and (2) $\vec{\alpha}$ self-assembles C_i in its entirety before C_j if $i < j$. By the way we defined the tile types of $T_{M(w)}$, it is easy to see that every tile that binds in $\vec{\alpha}$ does so deterministically, and with exactly strength-2 (either one strength-2 bond or two strength-1 bonds), whence $\mathcal{T}_{M(w)}$ is locally deterministic. The lemma follows by the addition of two tile types, each having strength-2 glues on their south edges, labeled with "accept" and "reject" respectively. □

The above "wedge" construction can be used to prove the following undecidability result.

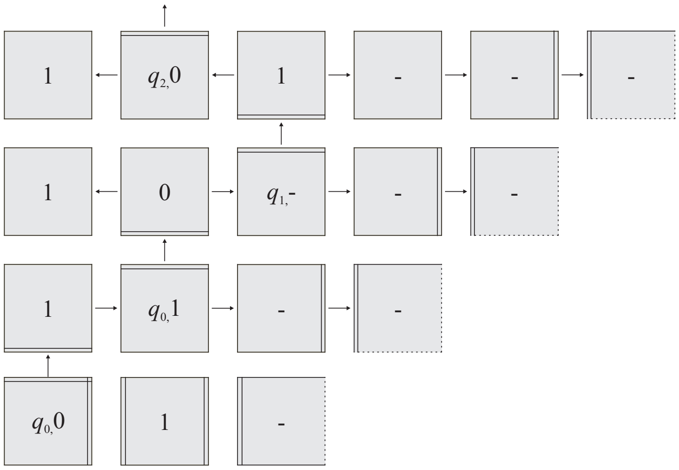


Figure 2: Example of the first four rows of a sample wedge construction which is simulating a Turing machine M on the input string '01'

Corollary 3.2. The language defined as

$$A = \{\mathcal{T} \mid \mathcal{T} \text{ is a TAS such that } \mathcal{T} \text{ is locally deterministic}\}$$

is undecidable.

Proof. We will prove the stronger result: A is Π^1_0-complete. To see that $A \in \Pi^1_0$, first note that

$$A = \left\{ \mathcal{T} \;\middle|\; \begin{array}{l} \text{for each } n \in \mathbb{N}, \vec{\alpha} \text{ is an assembly sequence in } \mathcal{T} \text{ of length } n, \\ \text{and } \vec{\alpha} \text{ satisfies the first two conditions of local determinism} \end{array} \right\}.$$

Checking whether a finite assembly sequence in some tile assembly sequence satisfies the first two conditions of local determinism is a decidable condition, whence $A \in \Pi^1_0$.

Next, to show that A is Π^1_0-hard, we will exhibit a many-one reduction from the complement of the halting problem H^c to A. Our reduction F takes as input a Turing machine M and a binary string $w \in \{0,1\}^*$, and outputs the tile assembly system $\mathcal{T}_{M(w)}$ modified as follows: each "final halting tile type" is replaced by two distinct tile types that each have a strength-2 glue on their south edge but share the same glue label (either "accept" or "reject"). This clearly breaks the second condition of local determinism! Note that we could also modify $\mathcal{T}_{M(w)}$ such that the final halting tiles bind with strength $3 > \tau = 2$, thus breaking the first condition of local determinism. It is clear that F is a reduction, seeing as how if M never halts on w, then $\mathcal{T}_{M(w)}$ remains locally deterministic (because no halting tiles ever attach). However, if M halts on w, the one of the newly added halting tiles will non-deterministically attach breaking the local determinism of $\mathcal{T}_{M(w)}$. □

4 A New Characterization of Decidable Languages

We now turn our attention to the self-assembly of decidable sets of positive integers. We will extend the wedge construction from the previous section in order to prove that, for every decidable set $A \subseteq \mathbb{N}$, there exists a directed TAS $\mathcal{T}_A = (T_A, \sigma, \tau)$ in which $A \times -\mathbb{N}$ weakly self-assembles. Our proof relies on the following observation.

Observation 1. *If $A \subseteq \mathbb{N}$ is a decidable set, then there is a TM M, such that for every $x \in \{0,1\}^*$, M halts on x.*

Observation 1 essentially says that we can simply "stack" wedge constructions one on top of the other. Intuitively, our main construction is the "self-assembly version" of the following enumerator.

```
while $1 \leq n < \infty$ do
  simulate $M$ on the binary representation of $n$
  if $M$ accepts then
    output 1
  else
    output 0
  end if
  $n := n + 1$
end while
```

Just as the above enumerator prints the characteristic sequence of A, our construction will self-assemble a canonical two-dimensional representation of the characteristic sequence of A as points along the negative y-axis.

4.1 Main Construction

In this section we present the full construction of the tile assembly system $\mathcal{T}_A$, and in the next section we provide a higher level description of the behavior of our tile system. Note that we used the Tile Set Designer graphical interface to the TAM DSL [1] to design the tile set for this construction and the ISU TAS simulator [4] to verify its correctness. Both software packages are available for download from `http://www.cs.iastate.edu/~lnsa`.

Lemma 4.1 (Main construction)**.** Let $A \subseteq \mathbb{N}$. There exists a directed, singly-seeded, tile assembly system $\mathcal{T}_A = (T_A, \sigma, 2)$ in which the set $A \times -\mathbb{N}$ weakly self-assembles.

Proof. Our proof is by construction. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ be a Turing machine with blank symbol '-' $\in \Gamma$ and $L(M) = A$. Assume, without loss of generality, that M is a total Turing machine having a one-way infinite-to-the-right tape such that the tape head of M never attempts to move left while reading the left most tape cell. We give the full specification of T_A and σ below.

Simulation tiles: Throughout our construction of simulation tiles, every tile takes as input, and ultimately outputs, six pieces of information along its south and north edges, respectively:

1. The symbol stored in the tape cell represented by this tile,
2. the current state of the Turing machine that is being simulated,
3. the direction of movement of the tape head,
4. the value of the bit that is embedded into this tile,
5. the significance of the aforementioned bit, and
6. a miscellaneous signal.

This situation is illustrated in Figure 3. Note that some or all of these six pieces of information might not be relevant at certain times during the assembly process. Therefore, we use underscores (i.e., "place holder" values) to denote the absence of values of some of the signals when they are not required. The following list of tile types encode the logic of the Turing machine M that decides A.

1. The following tile types appear only near the seed tile type (the tile having the 'S' label).

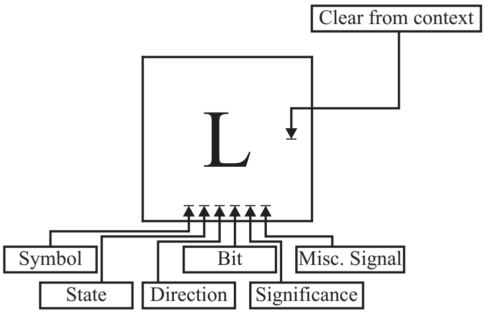
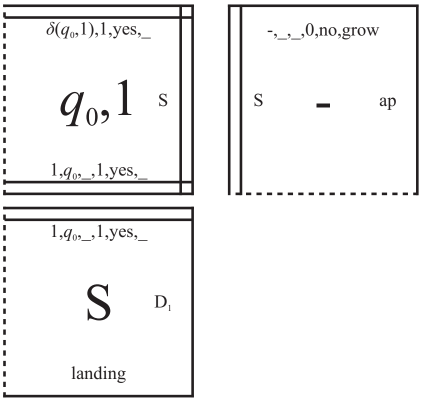
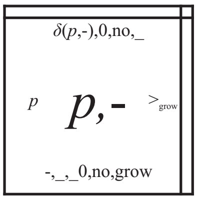


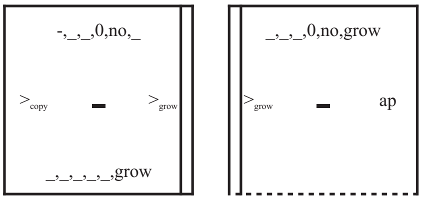
Figure 3: Each tile that contributes to the simulation of the Turing machine takes as input six pieces of information. We omit discussion of the east and west glue labels for our tile types because the *type* of information that is being taken as input and produced as output is always clear from the context.



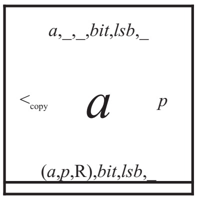
2. The following tile type only appears near the seed and receives the tape head from the left. For all $p \in Q - \{q_{\text{accept}}, q_{\text{reject}}\}$, add the following tile type.



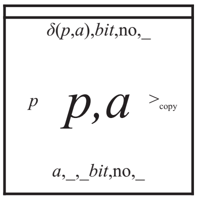
3. The following tile types “grow” the tape one cell to the right.



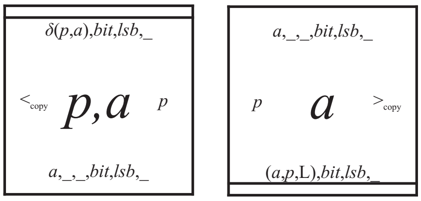
4. The following tile types move the tape head right. For all For all $p \in Q - \{q_{\text{accept}}, q_{\text{reject}}\}$, $a \in \Gamma$, $bit \in \{0, 1\}$, and $lsb \in \{\text{yes}, \text{no}\}$, add the following tile types. As noted above, the three pieces of information passed upward are the current symbol in this particular tape cell, and the current value of the bit of this column along with its significance.



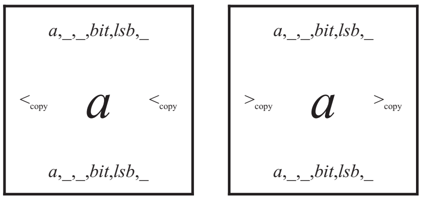
5. The following tile type receives the tape head from the left. For all For all $p \in Q - \{q_{\text{accept}}, q_{\text{reject}}\}$, $a \in \Gamma$, and $bit \in \{0, 1\}$, add the following tile types.



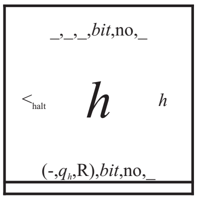
6. The following tile types receive the tape head from the right and move the tape head left (respectively). For all For all $p \in Q - \{q_{\text{accept}}, q_{\text{reject}}\}$, $a \in \Gamma$, $bit \in \{0, 1\}$, and $lsb \in \{\text{yes}, \text{no}\}$, add the following tile types.



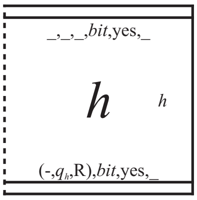
7. The following tile types copy the contents of the tape to the left and right of the tape head up to the next row (respectively). For all $a \in \Gamma$, $bit \in \{0, 1\}$, $lsb \in \{\text{yes}, \text{no}\}$, add the following tile types.



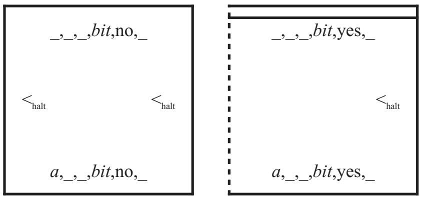
8. The following tile type halts the Turing machine when the tape head is not reading the left most tape cell. For all $h \in \{\text{accept}, \text{reject}\}$, and $bit \in \{0, 1\}$, add the following tile type.



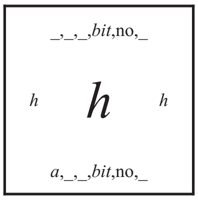
9. The following tile type halts the Turing machine when the tape head *is* reading the left most tape cell. For all $h \in \{\text{accept}, \text{reject}\}$, and $bit \in \{0, 1\}$, add the following tile type.



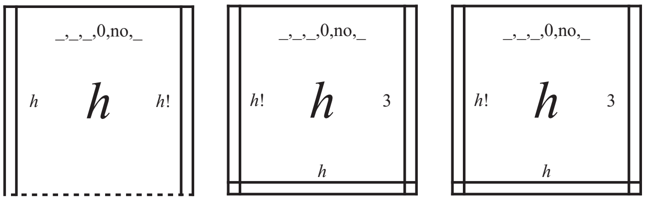
10. The following tile types search (to the left of the tape head) for the left most tape cell after halting. For all $a \in \Gamma$, and $bit \in \{0,1\}$, add the following tile types.



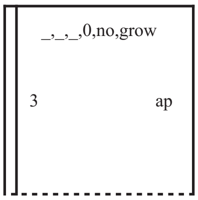
11. The following tile type transfers the halting state (either accept or reject) to the right enroute to the negative y-axis. For all $a \in \Gamma$ and $bit \in \{0,1\}$, add the following tile type.



12. The following tile types prepare to send the one-tile-wide path containing the halting state (either accept or reject) down to the negative y-axis. For all $h \in \{q_{\text{accept}}, q_{\text{reject}}\}$, add the following tile types.

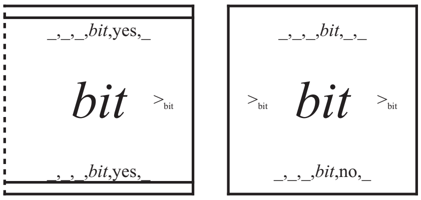


13. The following tile type is the right most tile to attach in any halting row. Add the following tile type.

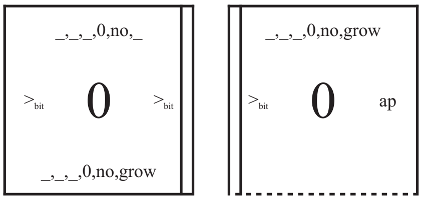


The following tile types "extract" the bits that are embedded within each simulation of the Turing machine. This is the final step before the count is incremented by one and used as input to the next simulation.

1. The following tile types initiate and carry out the bit extraction procedure starting from the least significant bit and working toward the right edge of the previous row of the assembly. For all $bit \in \{0,1\}$, add the following tile types.

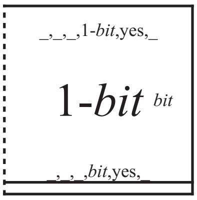


2. The following tile types extract the final two bits. The tile type on the left extracts the final bit of the previous row, and the tile type on the right side adds a dummy bit to maintain the geometry of the right most edge of the assembly. Add the following tile types.

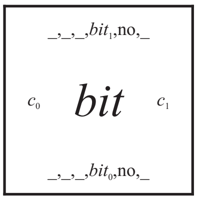


The following tile types self-assemble on top of the bit-extraction row and increments the value of these bits by 1.

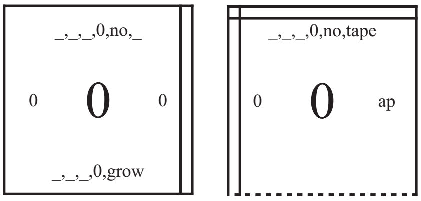
1. The following tile type initiates the increment process starting from the least significant bit. For all $bit \in \{0, 1\}$, add the following tile type.



2. The following tile type performs the bulk of the increment procedure. For all $bit_0 \in \{0, 1\}$ and $c_0 \in \{0, 1\}$, add the following tile type, where $c_1 = \frac{\lfloor bit_0 + c_0 \rfloor}{2}$ and $bit_1 = (bit_0 + c_0) \mod 2$.

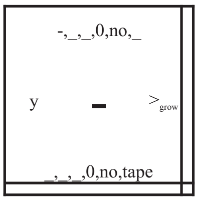


3. The following tile types are the final two tile types to attach in any increment row. Since the right edge of the construction grows faster than the length of the binary integers, the bits will always be 0. Add the following tile types.

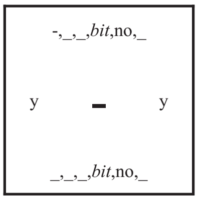


The following list of tile types build the initial configuration of the "next" simulation of M.

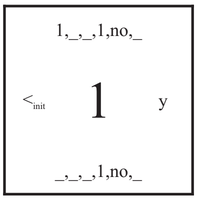
1. The following tile type assembles the right most tape cell (it always contains a blank) with a 0 bit embedded in it. Note that the 'y' signal is used to search for the right most 1 bit in the previous row. Add the following tile type



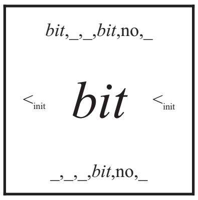
2. The following tile type continues the search (via the 'y' signal) for the right most 1 bit in the previous row. While doing so, blank symbols are encoded in the tape and 0 bits are also embedded. For all $bit \in \{0, 1\}$, add the following tile type.



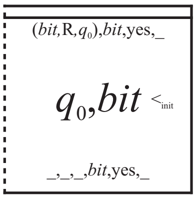
3. The following tile type attaches directly above the right most 1 bit in the previous row. In this case, the 1 bit is encoded in the tape and the embedded bit is 1. Note that this never occurs in the least significant bit. We terminate our search by changing the 'y' signal to $<_{\text{init}}$. Add the following tile types.



4. The following tile type initializes the contents of the tape to the left of the right most 1 bit. Here, we simply encode each cell with the appropriate bit from the previous row. Add the following tile type.

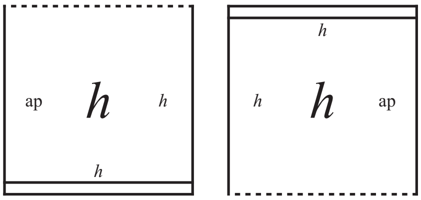


5. This tile type starts the next simulation of M. For all $bit \in \{0, 1\}$, add the following tile type.

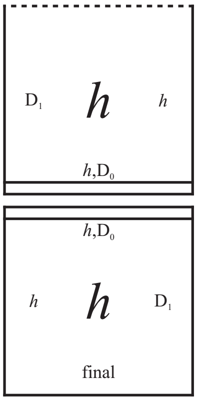


“Decision path” tiles: All of the previous tile types contribute to the simulation of M on every input $x \in \mathbb{N}$. We complete our construction by adding the tile types that self-assemble one-tile-wide “decision paths” that result in the placement of black (accept) or non-black (reject) tiles on the negative y-axis.

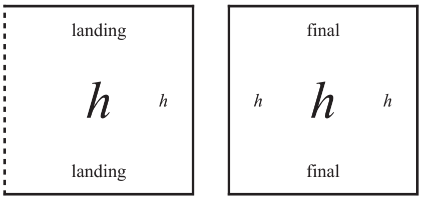
1. The following tile types initiate the one-tile-wide “decision path.” This path starts from the right edge of a halting row (see above tile types) and carries the answer to the question, “did the Turing machine accept or reject?” Note that the geometry of the existing assembly guides the path down to the appropriate point on the negative y-axis. For each $h \in \{\text{accept}, \text{reject}\}$, add the following tile types.



2. The following tile types allow the decision paths to “turn” the corner and thus head straight for the appropriate point on the negative y-axis. We use the D_0 and D_1 signals to accomplish this task. For each $h \in \{\text{accept}, \text{reject}\}$, add the following tile types.



3. The following tile types assemble the final segment of each decision path. The tile type that is placed on the negative y-axis is the left most tile below. For each $h \in \{\text{accept}, \text{reject}\}$, add the following tile types.



Let T_A be the set of *all* of the tile types that are defined above. Let σ be the seed assembly consisting of the unique tile type having the label ‘S’ placed at the origin and undefined at every other point $\vec{0} \neq \vec{x} \in \mathbb{Z}^2$. Finally, define the tile assembly system $\mathcal{T}_A = (T_A, \sigma, 2)$. A routine local determinism argument can be used to show that $\mathcal{T}_A$ is locally deterministic and therefore directed. Choosing the set B (a.k.a., the set of “black” tiles) to be the singleton set containing the left most tile type in the last pair of tile types defined above, where $h = \text{accept}$, proves that $A \times -\mathbb{N}$ weakly self-assembles. □

4.2 Discussion of Proof of Lemma 4.1

This section gives a high level, intuitive description of the constructive proof of Lemma 4.1. Note that $\mathcal{T}_A$ is a singly-seeded, two-dimensional temperature 2 tile assembly system. The seed tile type is the unique tile type having a label of 'S.' We place the seed tile at the point $(0, 0)$.

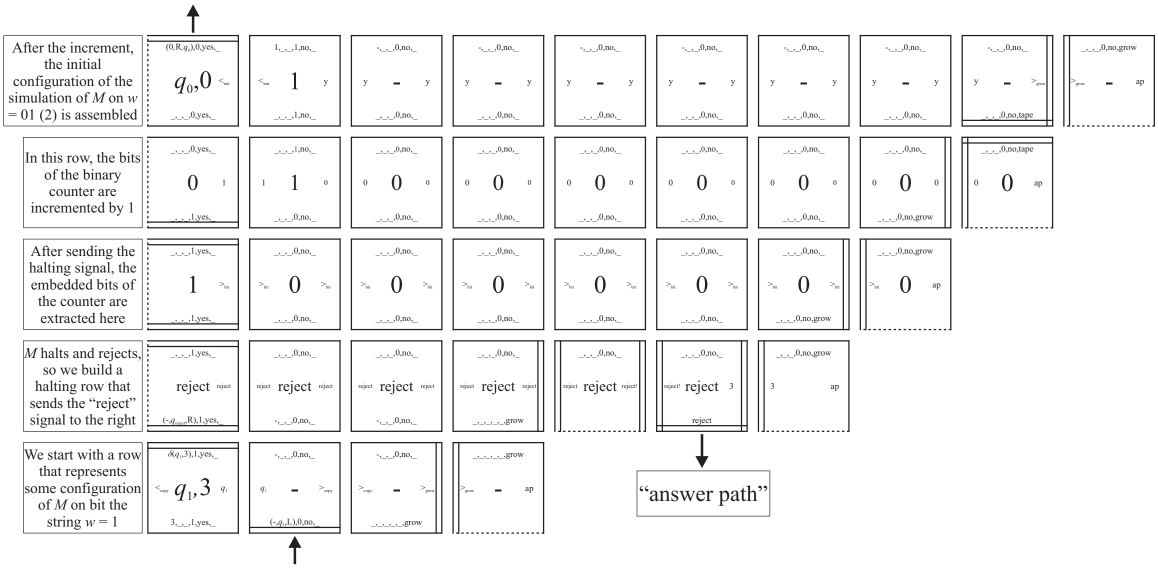

After the increment, the initial configuration of the simulation of M on w = 01 (2) is assembled
In this row, the bits of the binary counter are incremented by 1
After sending the halting signal, the embedded bits of the counter are extracted here
M halts and rejects, so we build a halting row that sends the "reject" signal to the right
We start with a row that represents some configuration of M on bit the string w = 1
q0,0
1
0
reject
q1,3
"answer path"
ap

Figure 4: Trivial example of the "halt-extract-increment-initialize" procedure carried out by our construction.

Definition 4.2. In our construction, for all $n \in \mathbb{N}$, there exists a one-tile-wide path that carries the answer to the question "is $n \in A$?" to the point $(0, -n)$. We call such a path a *decision path*.

The tile assembly system $\mathcal{T}_A$ consists of two logical modules. The first of these modules carries out the simulation of M on the binary representation of $1 \leq n \in \mathbb{N}$. In order to simulate M on the binary representation of every natural number, we embed a kind of log width binary counter into the tiles of $\mathcal{T}_A$. Thus, each tile must "remember" (and possibly modify) the value and significance of a particular bit in the binary counter. Note that, because of the way we embedded the counter, $\mathcal{T}_A$ actually simulates M on *the reverse of* the binary representation of every natural number. Tiles must also keep track of information pertaining to the simulation of M on a given input. The information that each tile is responsible for is shown visually in Figure 3.

The assembly process starts by $\mathcal{T}_A$ simulating M on the binary representation of 1. In general, after the simulation of M on i but *before* the simulation of M on $i + 1$, $\mathcal{T}_A$ performs the following tasks.

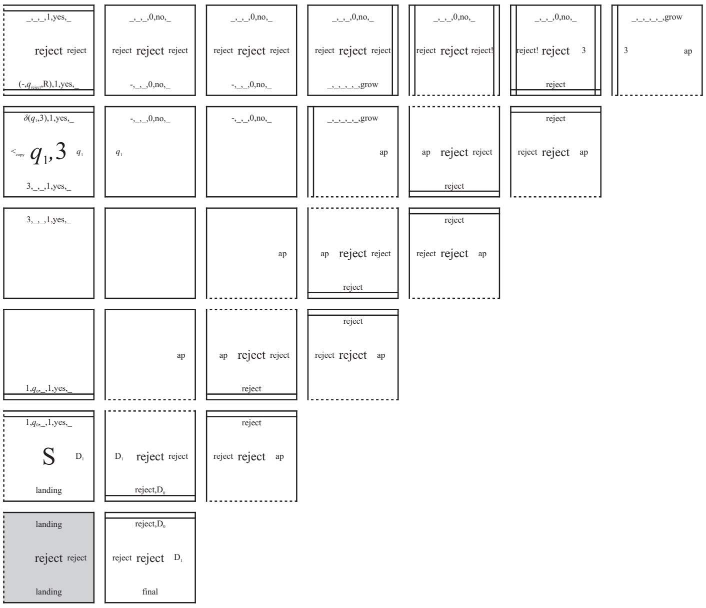
1. The answer to the question "Does M accept or reject i?" is propagated via a one-tile-wide "decision path" down to the point $(0, -i)$ (discussed below),

2. the bits of the embedded binary counter are extracted in the row immediately above the row of tiles representing the halting configuration of M on i,

3. in the row immediately above the row in which the bits of the binary counter were extracted, the value of the binary counter is incremented by 1, and finally,

4. the simulation of M on $i + 1$ is initialized and proceeds in the same fashion as that of M on i.

We give a concrete example of this four-step procedure in Figure 4.



This is the black tile placed at (0,-1) representing the fact that M rejects input $w = 1$

Figure 5: Trivial example of the one-tile-wide "decision path" that carries the answer to the question "is $1 \in L(M)$?" down to the appropriate point on the negative y-axis. Note that the D_0 and D_1 signals allow each decision path to turn the "corner" and proceed left toward the negative y-axis.

The second component of $\mathcal{T}_A$ is a small group of tile types that carry the "accept" and "reject" signals to the appropriate location on the negative y-axis via a one-tile-wide path of tiles. The geometry of the existing assembly "guides" these decision paths to the correct location. Each decision path originates from the halting tile and proceeds down toward the x-axis. In order for each path to turn the corner and proceed toward their final destination somewhere on the negative y-axis, we propagate "diagonal" signals (i.e., D_0 and D_1) down and to the right into the fourth quadrant. An example of this process is illustrated in Figure 5.

Note that the simulation component of $\mathcal{T}_A$ can self-assemble in the absence of the decision path component whereas the latter requires the former to self-assemble. Figure 6 shows the "flow" of information from the simulation components (the light grey spaces) to their respective halting rows (dark grey horizontal rows) down to the appropriate point on the negative y-axis via a decision path (the dark grey paths that snake down and to the left along the assembly).

4.3 First Main Theorem

The following technical result is a primitive self-assembly simulator.

Lemma 4.3. Let $A \subseteq \mathbb{Z}^2$. If A weakly self-assembles, then there exists a TM M_A with $L(M_A) = A$.

Proof. Assume that A weakly self-assembles. Then there exists a TAS $\mathcal{T} = (T, \sigma, \tau)$ in which the set A weakly self-assembles. Let B be the set of "black" tile types given in the definition of weak self-assembly. Fix some enumeration $\vec{a}_1, \vec{a}_2, \vec{a}_3 \dots$ of $\mathbb{Z}^2$, and let M_A be the TM, defined as follows.

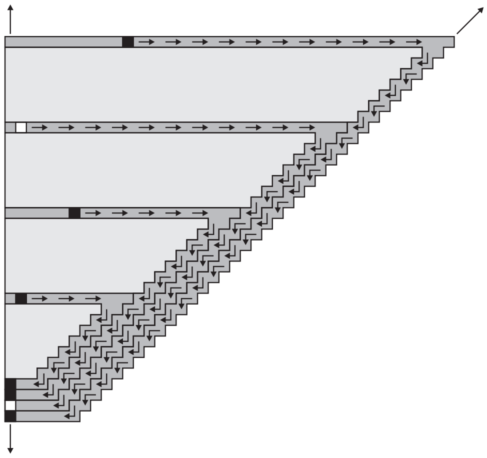


Figure 6: Intuitive depiction of the behavior of $\mathcal{T}_A$. The dark grey (horizontal) rows are halting rows. The arrows represent one-tile-wide paths of tiles that carry the answer to the question "is $n \in A$?" down to the negative y-axis.

```
Require: $\vec{v} \in \mathbb{Z}^2$
  $\alpha := \sigma$
  while $\vec{v} \notin \mathrm{dom}\, \alpha$ do
    choose the least $j \in \mathbb{N}$ such that some tile can be added to $\alpha$ at $\vec{a}_j$
    choose some $t \in T$ that can be added to $\alpha$ at $\vec{a}_j$
    add $t$ to $\alpha$ at $\vec{a}_j$
  end while
  if $\alpha(\vec{v}) \in B$ then
    accept
  else
    reject
  end if
```

It is routine to verify that M_A accepts A. □

Lemma 4.4. Let $A \subseteq \mathbb{N}$. If $A \times -\mathbb{N}$ and $A^c \times -\mathbb{N}$ weakly self-assemble, then A is decidable.

Proof. Assume the hypothesis. Then by Lemma 4.3, there exist TMs $M_{A\times\{0\}}$ and $M_{A^c\times-\mathbb{N}}$ satisfying $L(M_{A\times-\mathbb{N}}) = A \times -\mathbb{N}$, and $L(M_{A^c\times-\mathbb{N}}) = A^c \times -\mathbb{N}$, respectively. Now define the TM M as follows.

```
Require: $n \in \mathbb{N}$
  Simulate both $M_{A\times-\mathbb{N}}$ and $M_{A^c\times-\mathbb{N}}$ on input $(0, -n)$ in parallel.
  if $M_{A\times-\mathbb{N}}$ accepts then
    accept
  end if
  if $M_{A^c\times-\mathbb{N}}$ accepts then
    reject
  end if
```

It is clear that M is a decider for A. □

Lemma 4.5. Let $A \subseteq \mathbb{N}$. If the set A is decidable, then $A \times -\mathbb{N}$ and $A^c \times -\mathbb{N}$ weakly self-assemble.

Proof. This follows immediately from Lemma 4.1. □

We now have the machinery to prove our main result.

Theorem 4.6 (first main theorem)**.** Let $A \subseteq \mathbb{N}$. The set A is decidable if and only if $A \times -\mathbb{N}$ and $A^c \times -\mathbb{N}$ weakly self-assemble.

Proof. This follows from Lemmas 4.4 and 4.5. □

In the next section, we will analyze the space requirements of this assembly and present a series of constructions which require less space, but at a price of increasing the tile set (a.k.a., program-size [6]) complexity.

5 Space Requirements of the Self-Assembly of Decidable Sets

In the proof of Theorem 4.6, we exhibited a directed TAS whose (infinite) terminal assembly requires two full quadrants of the plane. This leads one to ask the natural question: is it possible to do any better than two quadrants? In other words, does Theorem 4.6 hold if only one quadrant of space, or less, is allowed? By investigating this, we hope to develop interesting "tile programming tricks".

If $A \in \text{DSPACE}(n)$, then it is possible to modify our construction to weakly self-assemble $A \times \{0\}$ using only one quadrant of space by making the tape for each computation $M(n)$ exactly $n-1$ tiles wide, then self-assembling a path with the result directly down the right side of the assembly to the location $(n, 0)$. However, in the case that $A \notin \text{DSPACE}(O(n))$, this modified construction does not suffice to self-assemble $A \times \{0\}$ within only one quadrant. In the remainder of this section we will discuss why this is the case, as well as alternative constructions which do suffice even for such complex languages, and their space requirements.

5.1 The Impact of the Decision Paths

It is clear that the portions of the assembly of our main construction, which initiate and perform each of the infinite series of computations, are contained within a wedge-shaped portion of a single quadrant. However, the decision paths end up being the portion of the assembly which force the additional space requirement of a second quadrant for sufficiently complex languages.

If the language $A \notin \text{DSPACE}(O(n))$, then there exists some n for which the computation $M(n)$ requires more than n tape cells, and therefore the assembly simulating $M(n)$ requires at least one row of tiles whose width is greater than n tiles. Since the row of tiles forming the computation will already be in place and cannot be removed to allow the decision path through, in order for the decision path emanating from this computation to terminate in the correct location, $(n, 0)$, it will have to go around that row, passing through an x-coordinate greater than n, then turn left at some point. (This ignores the possibility of the decision paths traveling down the left side of the assembly since they would therefore already be using a second quadrant.) Additionally, each of the infinite number of subsequent decision paths will then have to assemble around that path. Figure 7 depicts the situation where the first three paths in an infinite series of paths make a left turn. Note that each subsequent path must be translated one additional coordinate downward, and therefore, since there must be an infinite number of subsequent paths, eventually the decision path for some $m \in \mathbb{N}$ must place a tile on the x-axis in a location other than $(m, 0)$, or otherwise block another decision path. It is for this reason that the Main Construction requires the use of two quadrants for a language A of sufficient space complexity.

5.2 Squeezing More Information Into Each Decision Path

As shown above, the decision paths of the Main Construction form a virtual bottleneck which force the assembly into a second quadrant. This is due to the fact that each computation has its own, unique, decision path. However, if we "compress" more information into the decision paths, we can further reduce the space requirements. In fact, with a tradeoff in tile set complexity, where complexity is measured as the size of the tile set required, not only can one quadrant be made sufficient, but an arbitrarily small slice of a single quadrant can suffice.

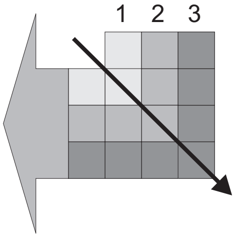


Figure 7: Example of a series of paths which make a left turn. Notice that if path "2" assembles after path "1", in order to make the turn it must be translated one additional coordinate downward. This must occur for each of the infinite series of paths which follow.

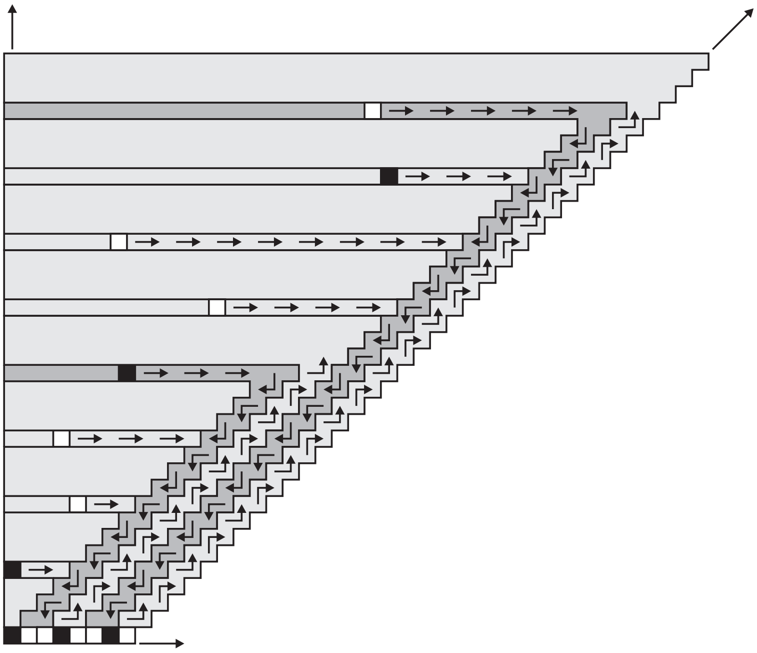


Figure 8: The single quadrant construction.

5.2.1 A Single-Quadrant Construction

For our next construction, instead of creating a decision path for each individual computation, we let four computations complete (i.e. $M(n)$, $M(n+1)$, $M(n+2)$, and $M(n+3)$) and then create a single decision path which collects and transmits all four answers to their correct locations on the x-axis. Figure 8 shows a high-level overview of this construction.

Essentially, an additional counter value which cycles from 0 to 3 and increments at the completion of each computation is passed upward through the right side of each computation. This allows the completion of each fourth computation to initiate the growth of a decision path which moves downward, collecting the previous three results until the single path contains four results. It then continues to the x-axis where it deposits all four results and initiates the growth of an upward growing path which eventually allows the next set of results to be propagated downward. Note that a main function of the upward growing path is to propagate a marker denoting which row is immediately above the x-axis to the right side of the assembly, and thus signal the next downward growing path when it needs to "unpack" its results.

This "compression" of four results into each result path obviates the need for the assembly to grow into a second quadrant, and it does so with only a trivial increase in tile set complexity, which comes mostly from the $2^2 + 2^3 + 2^4$ tile types which must be added to allow the collection and transmission of 4 results in each decision path.

5.2.2 Constructions Using Arbitrarily Thin "Pie" Slices of One Quadrant

We now demonstrate how we can further exploit the method of compressing increasingly larger sets of results into each decision path to create constructions that use arbitrarily thin (but infinite) pie-shaped slices of a single quadrant. These constructions will be accompanied by a quickly growing trade off in tile set complexity which we will also analyze. Note that we merely sketch these constructions and leave open the problem of implementation, and perhaps further optimization with regards to space and tile set complexity.

First, note that a standard wedge construction can be modified so that the leftmost tile of each row is s positions further to the right than the leftmost tile of the row immediately below it. This is done by:

1. Initiating the growth of each row at the leftmost position and growing from left to right.
2. Within each tile representing the nth tape cell from the left of the tape, encode the contents of tape cells $(n - s, \ldots, n)$, including the state information if one of those cells contains the tape head.

This works because the tile representing the tape cell that must receive the tape head after a left or right moving transition always assembles either directly above or above and to the right of the tile representing the tape head in the previous row, and therefore that information is available as necessary to each proceeding row. One additional difference from the standard wedge construction is that, if a computation enters a halting state while the tape head is on the n^{th} tape cell from the left, n/s additional rows (in which no computation is performed) will need to assemble before the information that it halted reaches the leftmost edge and the assembly of that simulation can terminate.

By using such a "slanted" wedge construction in the construction of Section 5.2.1 and modifying the decision paths to conform to the desired slope and to accumulate and carry $2s + 4$ results to the x-axis, (along with a few other trivial modifications) an assembly which requires only the portion of the first quadrant lying below the line $y = x/s$ can form which weakly self-assembles $A \times \{0\}$. Two examples of such constructions and their desired behaviors can be seen in Figure 9.

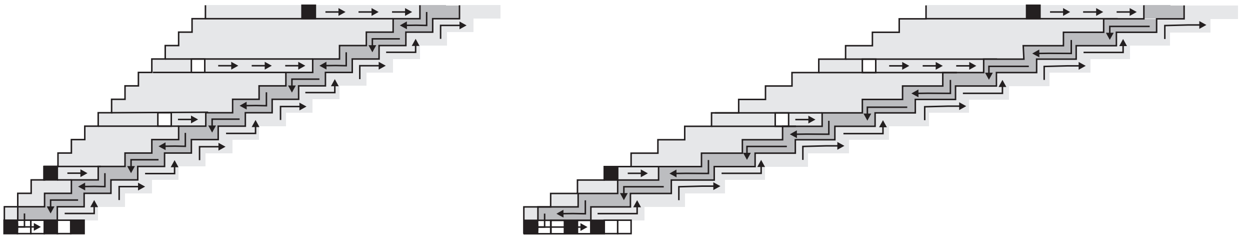


Figure 9: Examples of assemblies which weakly self-assembly the set $A \times \{0\}$ within the portions of Quadrant 1 (approximately) underneath the lines $y = x$ (left) and $y = x/2$ (right), corresponding to $s = 1$ and $s = 2$, respectively. Note that the constructions must combine 6 (for the left) and 8 (for the right) signals into each decision path.

We conjecture that the tradeoff for shrinking the assembly into such arbitrarily small "pie slices" of the first quadrant is an exponential increase in tile set complexity. In fact, for a given s, the size of the tile set required for our proposed assembly would be $O(\gamma^s)$ where $\gamma = |\Gamma|$ (and Γ is the tape alphabet of the Turing machine M being simulated).

6 Conclusion

In this paper, we investigated the self-assembly of decidable sets of natural numbers in the TAM. We proved that, for every decidable language $A \subseteq \mathbb{N}$, $A \times \{0\}$ and $A^c \times \{0\}$ weakly self-assemble. This implied a novel characterization of decidable sets in terms of self-assembly. We then analyzed the space requirements of our construction and showed that, in general, for decidable languages, our construction requires two quadrants of space. This led us to the presentation of slightly modified constructions which required, first, only one quadrant of space, and then increasingly smaller portions of a single quadrant. However, this decrease in space was accompanied a "proportional" increase in the size of the tile set complexity.

Additional examples of the trade off between the space complexity of languages and the amount of space required to self-assemble their characteristic sequences in the TAM include the fact that one *spatial* dimension is sufficient to self-assemble $A \times \{0\}$ if and only if A is a regular language over a unary alphabet, and our speculation that if A is a regular language over a binary alphabet that only one *fractal* dimension would be required. Many open problems related to such relationships remain, such as the implementation and potential optimization of our "pie-slice" constructions, and a proof of our above speculation about regular languages over binary alphabets. We hope that continued research in this direction will further extend these results, exposing, more and more, the rich interconnectedness between geometry and computation in the TAM.

Acknowledgment

Both authors wish to thank David Doty, Jack Lutz and Damien Woods for useful discussions.

References

[1] David Doty and Matthew J. Patitz, *A domain specific language for programming in the tile assembly model*, Proceedings of The Fifteenth International Meeting on DNA Computing and Molecular Programming (Fayetteville, Arkansas, USA, June 8-11, 2009), 2009, to appear.

[2] James I. Lathrop, Jack H. Lutz, Matthew J. Patitz, and Scott M. Summers, *Computability and complexity in self-assembly*, Proceedings of The Fourth Conference on Computability in Europe (Athens, Greece, June 15-20, 2008), 2008.

[3] James I. Lathrop, Jack H. Lutz, and Scott M. Summers, *Strict self-assembly of discrete Sierpinski triangles*, Proceedings of The Third Conference on Computability in Europe (Siena, Italy, June 18-23, 2007), 2007.

[4] Matthew J. Patitz, *Simulation of self-assembly in the abstract tile assembly model with ISU TAS*, 6th Annual Conference on Foundations of Nanoscience: Self-Assembled Architectures and Devices (Snowbird, Utah, USA, April 20-24 2009). To appear., 2009.

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

[6] 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.

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

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

[9] ______, *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.

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