Intrinsic Universality in Self-Assembly

David Doty, Jack H. Lutz, Matthew J. Patitz, Scott M. Summers, Damien Woods

HAL Id: inria-00455225
https://inria.hal.science/inria-00455225

Submitted on 9 Feb 2010

INTRINSIC UNIVERSALITY IN SELF-ASSEMBLY

DAVID DOTY [1] AND JACK H. LUTZ [2] AND MATTHEW J. PATITZ [2] AND SCOTT M. SUMMERS [2] AND DAMIEN WOODS [3]

[1] Department of Computer Science, University of Western Ontario
London, Ontario, Canada, N6A5B7.
E-mail address, David Doty: `ddoty@csd.uwo.ca`

[2] Department of Computer Science, Iowa State University
Ames, IA 50011 USA.
E-mail address, Jack H. Lutz: `lutz@cs.iastate.edu`
E-mail address, Matthew J. Patitz: `patitz@cs.iastate.edu`
E-mail address, Scott M. Summers: `summers@cs.iastate.edu`

[3] California Institute of Technology, Pasadena, CA 91125, USA.
E-mail address: `woods@caltech.edu`

ABSTRACT. We show that the Tile Assembly Model exhibits a strong notion of universality where the goal is to give a single tile assembly system that simulates the behavior of any other tile assembly system. We give a tile assembly system that is capable of simulating a very wide class of tile systems, including itself. Specifically, we give a tile set that simulates the assembly of any tile assembly system in a class of systems that we call *locally consistent*: each tile binds with exactly the strength needed to stay attached, and that there are no glue mismatches between tiles in any produced assembly.

Our construction is reminiscent of the studies of *intrinsic universality* of cellular automata by Ollinger and others, in the sense that our simulation of a tile system T by a tile system U represents each tile in an assembly produced by T by a $c \times c$ block of tiles in U, where c is a constant depending on T but not on the size of the assembly T produces (which may in fact be infinite). Also, our construction improves on earlier simulations of tile assembly systems by other tile assembly systems (in particular, those of Soloveichik and Winfree, and of Demaine et al.) in that we simulate the actual process of self-assembly, not just the end result, as in Soloveichik and Winfree's construction, and we do not discriminate against infinite structures. Both previous results simulate only temperature 1 systems, whereas our construction simulates tile assembly systems operating at temperature 2.

SYMPOSIUM
ON THEORETICAL
ASPECTS
OF COMPUTER
SCIENCE

1. Introduction

The development of DNA tile self-assembly has moved nanotechnology closer to the goal of engineering useful systems that assemble themselves from molecular components. Since Seeman's pioneering work in the 1980s [21], many laboratory experiments have shown that DNA tiles can be designed to spontaneously assemble with one another into desired structures [20]. As physical and mathematical error-suppression techniques improve [3, 8, 13, 22, 24], this molecular programming of matter will become practical at ever larger scales.

The Tile Assembly Model, developed by Winfree [19, 26], is a discrete mathematical model of DNA tile self-assembly that enables us to explore the potentialities and limitations of this kind of molecular programming. It is essentially an "effectivization" of classical Wang tiling [25] in which the fundamental components are un-rotatable, but translatable square "tile types" whose sides are labeled with glue "colors" and "strengths." 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 with total strength at least a certain ambient "temperature." Extensive refinements of the abstract Tile Assembly Model were given by Rothemund and Winfree in [18,19]. (Consult the technical appendix for full details of the abstract Tile Assembly Model.) The model deliberately oversimplifies the physical realities of self-assembly, but Winfree proved that it is Turing universal [26], implying that self-assembly can be algorithmically directed.

In this paper we investigate whether the Tile Assembly Model is capable of a much stronger notion of universality where the goal is to give a single tile assembly system that simulates the behavior of any other tile assembly system. We give a tile assembly system that is capable of simulating a very wide class of tile systems, including itself. Our notion of simulation is inspired by, but somewhat stronger than, intrinsic universality in cellular automata [2, 7, 14–16]. In our construction a simulated tile assembly system is encoded in a seed assembly of the simulating system. This encoding is done in a very simple (logspace computable) way. The seed assembly then grows to form an assembly that is a re-scaled (larger) version of the simulated assembly, where each tile in the latter is represented by a supertile (square of tiles) in the simulator. Not only this, but each of the possible (nondeterministically chosen) assembly sequences of the simulated tile system is modeled by a possible assembly sequence in the simulating system (also nondeterministically chosen). The latter property of our system is important and highlights one way in which this work distinguishes itself from other notions of intrinsic universality found in the cellular automata literature: not only do we want to simulate the final assembly but we also want the simulator to have the ability to dynamically simulate each of the valid growth processes that could lead to that final assembly.

A second distinguishing property of our universal tile set is that it simulates nondeterministic choice in a "fair" way. An inherent feature of the Tile Assembly Model is the fact there are often multiple (say k) tiles that can go into any one position in an assembly sequence, and one of these k is nondeterministically chosen. One way to simulate this feature is to nondeterministically choose which of k supertiles should grow in the analogous (simulated) position. However, due to the size blowup in supertiles caused by encoding an arbitrary-sized simulated tile set into a fixed-sized universal tile set, it seems that we need to simulate one nondeterministic choice by using a sequence of nondeterministic choices within the supertile. Interpreting the nondeterministic choice to be made according to uniform random selection, if the selection by the simulating tile set is implemented in a naïve

way, this can lead to unfair selection: when selecting 1 supertile out of k, some supertiles are selected with extremely low probability. To get around this problem, our system uses a random number selector that chooses a random tile with probability $\Theta(1/k)$ and so we claim that we are simulating nondeterminism in a "fair" way.

Thirdly, the Tile Assembly Model has certain geometric constraints that are not seen in cellular automata, and this adds some difficulty to our construction. Existing techniques for constructing intrinsically universal cellular automata are not directly applicable to tile assembly. For example, when a tile is placed at a position, that position can not be reused for further "computation" and this presents substantial difficulties when trying to fit the various components of our construction into a supertile. Each supertile encodes the entire simulated tile set and has the functionality to propagate this information to other (yet to be formed) supertiles. Not only this, each supertile must decide which tile placement to simulate, whilst making (fair) nondeterministic choices if necessary. Finally, each supertile should correctly propagate (output) sides that are consistent with the chosen supertile. We give a number of figures to illustrate how these goals were met within the geometric constraints of the model.

Our main result presented in this paper is, in some sense, a continuation of some previous results in self-assembly. For instance, Soloveichik and Winfree [23] exhibit a beautiful connection between the Kolmogorov complexity of a finite shape X and the minimum number of tiles types needed to assemble X. It turns out that their construction can be made to be "universal" in the following sense: there exists a tile set T, such that for every "temperature 1" tile assembly system that produces a finite shape whose underlying binding graph is a spanning tree, T simulates the given temperature 1 tile system with a corresponding blow-up in the scale. Note that this method restricts the simulated tile system to be temperature 1, i.e., a non-cooperative tile assembly system, which are conjectured [6] to produce "simple" shapes and patterns in the sense of Presburger arithmetic [17].

A similar result, recently discovered by Demaine, Demaine, Fekete, Ishaque, Rafalin, Schweller, and Souvaine [4], established the existence of a general-purpose "staged-assembly" system that is capable of simulating any temperature 1 tile assembly system that produces a "fully connected" finite shape. Note that, in this construction, the scaling factor is proportional to $O(\log |T|)$, where T is the simulated tile set. This construction has the desirable property that the set of tile types belonging to the simulator is general purpose (i.e., the size of the simulator tile set is independent of the to-be-simulated tile set) and all of the information needed to carry out the simulation is, in some sense, encoded in a sequence of laboratory steps. An open question in [4] is whether or not their construction can be augmented to handle temperature 2 tile assembly systems.

Our construction is general enough to be able to simulate powerful and interesting tile sets, yet sufficiently simple so that it actually belongs to the class of tile assembly systems that it can simulate, a class we term *locally consistent*. Systems in this class have the properties that each tile binds with exactly strength 2, and there are no glue mismatches in any producible assembly. This captures a wide class of tile assembly systems, including counters, square-builders and other shape-building tile assembly systems, and the tile assembly systems described in $[1, 12, 19, 23]$. Modulo re-scaling, our universal tile set can be said to display the characteristics of the entire collection of tile sets in its class. Our construction is a *direct simulation* in that the technique does not involve the simulation of intermediate models (such as circuits or Turing machines), which have been used in intrinsically universal cellular automata constructions [16].

One of the nice properties of intrinsic universality [16] is that it provides a clear definition that facilitates proofs that a given tile set is not universal. We leave as an open problem the intrinsic universality status of the Tile Assembly Model in its full generality.

Lafitte and Weiss [9–11] have also studied universality in the related model of Wang tiling [25]. Some of their definitions, particularly in [10], are similar to our definitions of simulation and universality, and also to those of Ollinger [16]. However, Wang tiling is not a model of self-assembly, as it is concerned with the ability of finite tile sets to tile the whole plane (with no mismatches), without regard to the *process* by which these tiles are placed. What is important is simply the existence of some valid tiling. In the TAM, which takes the order in which tiles are placed, one by one, into account, it must be shown that not only is there a sequence by which tiles could be individually and stably added to form the output assembly, but that *every* possible such sequence leads to the desired output. Furthermore, in the TAM a tile addition can be valid even if it causes mismatches as long as it is stable.

Most attempts to adapt the constructions of Wang tiling studies (such as those in [9–11]) to self-assembly result in a tile assembly system in which many junk assemblies are formed due to incorrect nondeterministic choices being made that arrest any further growth and/or result in assemblies which are inconsistent with the desired output assembly. We therefore require novel techniques to ensure that no nondeterminism is introduced, other than that already present in the tile system being simulated, and that the only produced assemblies are those that represent the intended result or valid partial progress toward it.

2. Intrinsic Universality in Self-Assembly

In this section, we define our notion of intrinsic universality of tile assembly systems. It is inspired by, but distinct from, similar notions for cellular automata [16]. Where appropriate, we identify where some part of our definition differs from the "corresponding" parts in [16], typically due to a fundamental difference between the abstract Tile Assembly Model and cellular automata models.

Intuitively, a tile set U is universal for a class $\mathfrak{C}$ of tile assembly systems if U can "simulate" any tile assembly system in $\mathfrak{C}$, where we use an appropriate seed assembly to give a tile assembly system $\mathcal{U}$. U is intrinsically universal if the simulation of $\mathcal{T}$ by $\mathcal{U}$ can be done according to a simple "block substitution scheme" where equal-size square blocks of tiles in assemblies produced by $\mathcal{U}$ represent tiles in assemblies produced by $\mathcal{T}$. Furthermore, since we wish to simulate the entire process of self-assembly, and not only the final result, it is critical that the simulation be such that the "local transition rules" involving intermediate producible (and nonterminal) assemblies of $\mathcal{T}$ be faithfully represented in the simulation.

In the subsequent definitions, given two partial functions f, g, we write $f(x) = g(x)$ if f and g are both defined and equal on x, or if f and g are both undefined on x. Let $c, c' \in \mathbb{N}$, let $[c : c']$ denote the set $\{c, c+1, \ldots, c'-1\}$, and let $[c]$ denote the set $[0 : c] = \{0, 1, \ldots, c-1\}$, so that $[c]^2$ forms a $c \times c$ square with the origin as the lower-left corner.

The natural analog of a configuration of a cellular automaton is an assembly of a tile assembly system. However, unlike cellular automata in which every cell has a well-defined state, in tile assembly, there is a fundamental difference between a point being empty space and being occupied by a tile. Therefore we keep the convention of representing an assembly as a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$ (for some tile set T), rather than treating empty space as just another type of tile.

Let $\mathcal{T} = (T, \sigma_{\mathcal{T}}, \tau)$ and $\mathcal{S} = (S, \sigma_{\mathcal{S}}, \tau)$ be tile assembly systems. For simplicity, assume that $\sigma_{\mathcal{T}}(0,0)$ is defined, and $\sigma_{\mathcal{T}}$ is undefined on $\mathbb{Z}^2 - \{(0,0)\}$ (i.e., $\mathcal{T}$ is singly-seeded with the seed tile placed at the origin). We will use this assumption of a single seed throughout the paper, but it is not strictly necessary and is only used for simplicity of discussion. Define a *representation function* to be a partial function of the form $r : ([c]^2 \dashrightarrow S) \dashrightarrow T$. That is, r takes a pattern $p : [c]^2 \dashrightarrow S$ of tile types from S painted onto a $c \times c$ square (with locations at which p is undefined representing empty space), and (if r is defined for input p) gives a single tile type from T. Intuitively, r tells us how to interpret $c \times c$ blocks within assemblies of $\mathcal{S}$ as single tiles of T. We write REPR for the set of all representation functions.

We say $\mathcal{S}$ *(intrinsically) simulates $\mathcal{T}$ with resolution loss c* if there exists a representation function $r : ([c]^2 \dashrightarrow S) \dashrightarrow T$ such that the following conditions hold.

(1) $\operatorname{dom} \sigma_{\mathcal{S}} \subseteq [c]^2$ and $r(\sigma_{\mathcal{S}}) = \sigma_{\mathcal{T}}(0,0)$, i.e., the seed assembly of $\mathcal{S}$ represents the seed of $\mathcal{T}$.

(2) For every producible assembly $\alpha_{\mathcal{T}} \in \mathcal{A}[\mathcal{T}]$ of $\mathcal{T}$, there is a producible assembly $\alpha_{\mathcal{S}} \in \mathcal{A}[\mathcal{S}]$ of $\mathcal{S}$ such that, for every $x, y \in \mathbb{Z}$,

$$r\left((\alpha_{\mathcal{S}} \upharpoonright ([cx : c(x+1)] \times [cy : c(y+1)])) + (-cx, -cy)\right) = \alpha_{\mathcal{T}}(x, y).$$

That is, the $c \times c$ block at (relative) position (x, y) (relative to the other $c \times c$ blocks; the absolute position is (cx, cy)) of assembly $\alpha_{\mathcal{S}}$ represents the tile type at (absolute) position (x, y) of assembly $\alpha_{\mathcal{T}}$. In this case, write $r^*(\alpha_{\mathcal{S}}) = \alpha_{\mathcal{T}}$; i.e., r induces a function $r^* : \mathcal{A}[\mathcal{S}] \to \mathcal{A}[\mathcal{T}]$.

(3) For all $\alpha_{\mathcal{T}}, \alpha'_{\mathcal{T}} \in \mathcal{A}[\mathcal{T}]$, it holds that $\alpha_{\mathcal{T}} \to_{\mathcal{T}} \alpha'_{\mathcal{T}}$ if and only if there exist $\alpha_{\mathcal{S}}, \alpha'_{\mathcal{S}} \in \mathcal{A}[\mathcal{S}]$ such that $r^*(\alpha_{\mathcal{S}}) = \alpha_{\mathcal{T}}$, $r^*(\alpha'_{\mathcal{S}}) = \alpha'_{\mathcal{T}}$ (in the sense of condition (2)), and $\alpha_{\mathcal{S}} \to_{\mathcal{S}} \alpha'_{\mathcal{S}}$. That is, every valid assembly sequence of $\mathcal{T}$ can be "mimicked" by $\mathcal{S}$, but no other assembly sequences can be so mimicked, so that the meaning of the relation $\to$ is preserved by r^*.

Let $\mathfrak{C}$ be a class of singly-seeded tile assembly systems, and let U be a tile set (with tile assembly systems having tile set U not necessarily elements of $\mathfrak{C}$). Note that every element of $\mathfrak{C}$, REPR, and $\operatorname{FIN}(U)$ is a finite object, hence can be represented in a suitable format for computation in some formal system such as Turing machines. We say U is *(intrinsically) universal* for $\mathfrak{C}$ if there are computable functions $R : \mathfrak{C} \longrightarrow$ REPR and $A : \mathfrak{C} \longrightarrow \operatorname{FIN}(U)$ such that, for each $\mathcal{T} = (T, \sigma_{\mathcal{T}}, \tau) \in \mathfrak{C}$, there is a constant $c \in \mathbb{N}$ such that, letting $r = R(\mathcal{T})$, $\sigma = A(\mathcal{T})$, and $\mathcal{U}_{\mathcal{T}} = (U, \sigma, \tau)$, $\mathcal{U}_{\mathcal{T}}$ simulates $\mathcal{T}$ with resolution loss c and representation function r. That is, $R(\mathcal{T})$ outputs a representation function that interprets assemblies of $\mathcal{U}_{\mathcal{T}}$ as assemblies of $\mathcal{T}$, and $A(\mathcal{T})$ outputs the seed assembly used to program tiles from U to represent the seed tile of $\mathcal{T}$.

3. An Intrinsically Universal Tile Set

In this section, we exhibit an intrinsically universal tile set for any "nice" tile assembly system. Before proceeding, we must first define the notion of a "nice" tile assembly system. Let $\mathcal{T} = (T, \sigma, 2)$ be a tile assembly system, and $\vec{\alpha}$ be an assembly sequence in $\mathcal{T}$ whose result is denoted as α. We say that $\mathcal{T}$ is *locally consistent* if the following conditions hold.

(1) For all $\vec{m} \in \operatorname{dom} \alpha - \operatorname{dom} \sigma$, $\sum_{\vec{u} \in \operatorname{IN}^{\vec{\alpha}}(\vec{m})} \operatorname{str}_{\alpha(\vec{m})}(\vec{u}) = 2$, where $\operatorname{IN}^{\vec{\alpha}}(\vec{m})$ is the set of sides on which the tile that $\vec{\alpha}$ places at location $\vec{m}$ initially binds. That is, every tile initially binds to the assembly with exactly bond strength equal to 2 (either a single strength 2 bond or two strength 1 bonds).

(2) For all producible assemblies $\alpha \in \mathcal{A}[\mathcal{T}]$, $\vec{u} \in U_2$, and $\vec{m} \in \text{dom } \alpha$, if $\alpha(\vec{m} + \vec{u})$ is defined, then the following condition holds:

$$\text{str}_{\alpha(\vec{m})}(\vec{u}) > 0 \Rightarrow \text{label}_{\alpha(\vec{m})}(\vec{u}) = \text{label}_{\alpha(\vec{m}+\vec{u})}(-\vec{u}) \text{ and } \text{str}_{\alpha(\vec{m})}(\vec{u}) = \text{str}_{\alpha(\vec{m}+\vec{u})}(-\vec{u}).$$

While condition (1) of the above definition is reminiscent of the first condition of local determinism [23], the second condition says that there are no (positive strength) label mismatches between abutting tiles. However, we must emphasize that a locally consistent tile assembly system need not be directed, and moreover, even a locally deterministic tile assembly system need not be locally consistent because of the lack of any kind of "determinism restriction" in the latter definition. Our main result is the following.

Theorem 3.1 (Main theorem). *Let $\mathfrak{C}$ be the set of all locally consistent tile assembly systems. There exists a finite tile set U that is intrinsically universal for $\mathfrak{C}$.*

In the remainder of this section, we prove Theorem 3.1, that is, we show that for every locally consistent tile assembly system $\mathcal{T} = (T, \sigma, 2)$, there exists a seed assembly $\sigma_\mathcal{T}$, such that the tile assembly system $\mathcal{U}_\mathcal{T} = (U, \sigma_\mathcal{T}, 2)$ simulates $\mathcal{T}$ with a resolution loss $c \in \mathbb{N}$ that depends only on the glue complexity of $\mathcal{T}$. Instead of giving an explicit (and tedious) definition of the tile types in U, we implicitly define U by describing how $\mathcal{U}_\mathcal{T}$ simulates $\mathcal{T}$.

3.1. High-Level Overview

Intuitively, $\mathcal{U}$ simulates $\mathcal{T}$ by growing "supertiles" that correspond to tile types in T. In other words, every supertile is a $c \times c$ block of tiles that is mapped to a tile type $t \in T$. To do this, each supertile that assembles in $\mathcal{U}_\mathcal{T}$ contains the full specification of T as a lookup table (a long row of tiles that encodes all of the information in the set of tile types T), analogous to the genome of an organism being fully replicated in each cell of that organism, no matter how specialized the function of the cell. This lookup table is carefully propagated through each supertile in $\mathcal{U}_\mathcal{T}$ via a series of "rotation" and "copy" operations – both of which are well-known self-assembly primitives.

In the table, we represent each (glue,direction) pair as a binary string, and represent the tile set as a table mapping 1-2 input glue(s) to 0-3 output glue(s). Since each tile type of $\mathcal{T}$ may not have well-defined input sides, when two supertiles representing tiles of $\mathcal{T}$ must potentially cooperate to place a new supertile within a block adjacent to both of them, it is imperative that each grows into the block in such a way as to remain unobtrusive to the other supertile. This is done with a "probe" that grows toward the center of the block, as shown in Figure 1. At the moment the probes meet in the middle, they "find out" in what direction the other input supertile lies, and at that point decide in which direction to grow the rest of the forming supertile. so as to avoid the tiles that were already placed as part of the probes. We do not know how to deal with three probes at once, which is the reason both parts of the definition of locally consistent, which imply that only two input probes will ever be present at one time. The next step is to bring the values of two input glues together before doing a lookup on the table, because they are both needed to simulate cooperation. The table must be read and copied at the same time, otherwise the planarity of the tiles would hide the table as it is read and it could not be propagated to the output supertiles. Many choices made in the construction, such as the relative positioning of glues/table, or the counter-clockwise order of assembly, are choices that simply were convenient and seemed to work, but are not necessarily required.

3.2. Construction of the Lookup Table

In order to simulate the behavior of $\mathcal{T}$ with $\mathcal{U}_\mathcal{T}$, we must first encode the definition of T using tiles from U. We will do this by constructing a "glue lookup table," denoted as $\mathbf{T}_\mathcal{T}$, and is essentially the self-assembly version of a kind of hash table. Informally, $\mathbf{T}_\mathcal{T}$ is a (very) long string (of tiles from U) consisting of two copies of the definition of the tile set T separated by a small group of spacer symbols. The left copy of the lookup table is the *reverse* of the right copy. The lookup table maps all possible sets of input sides for each tile type $t \in T$ to the corresponding sets of output sides.

3.2.1. *Addresses.* The lookup table $\mathbf{T}_\mathcal{T}$ consists of a contiguous sequence of "addresses," which are formed from the definition of T. Namely, for each tile type $t \in T$, we create a unique binary key for each combination of sides of t whose glue strengths sum to exactly 2. Each of these combinations represents a set of sides which *could* potentially serve as the input sides for a tile of type t in a producible assembly in $\mathcal{T}$.

We say that a *pad* is an ordered triple (g, d, s) where g is a glue label in T, $d \in \{N, S, E, W\}$ is an edge direction, and $s \in \{0, 1, 2\}$ is an allowable glue strength. Note that a set of four pads – one for each direction d – fully specifies a tile type. We use $\mathrm{Pad}(t, d)$ to denote the pad on side d of the tile type $t \in T$

Let $\mathrm{Bin}(p)$ be the binary encoding of a pad $p = (g, d, s)$, consisting of the concatenation of the following component binary strings:

(1) g (*glue specification*): Let G be the set of glue types from all edges with positive glue strengths in $T \cup \{g_{\mathrm{null}}\}$ (a.k.a., the null glue). Fix some ordering $g_{\mathrm{null}} \le g_0 \le g_1 \le \cdots$ of the set G. The binary representation of g_i is the binary value of i padded with 0's to the left (as necessary) to ensure that the string is exactly $\lceil \log(|G| + 1) \rceil$ bits.
(2) d (*direction*): If $d = N$ (E, S, or W), append 00 (01, 10, or 11, respectively).
(3) s (*strength*): If $s = 1$ (2) append 0 (1).

Note that $\lceil \log(|G| + 1) \rceil + 2 + 1$ is the length of the binary string encoding an arbitrary pad p, and is a constant that depends only on T.

An *address* is a binary string that represents a set of pads which, themselves, can potentially serve as the input sides of some tile type $t \in T$. It can be composed of one of the two following binary strings:

(1) A prefix of zeros, $0^{\lceil \log(|G| + 1) \rceil + 3}$, followed by $\mathrm{Bin}(p)$ for $p = (g, d, 2)$, or
(2) the concatenation of $\mathrm{Bin}(p_1)$ and $\mathrm{Bin}(p_2)$ for $p_1 = (g_1, d_1, 1)$ and $p_2 = (g_2, d_2, 1)$. The ordering of $\mathrm{Bin}(p_1)$ and $\mathrm{Bin}(p_2)$ in an address must be consistent with the following orderings: EN, SE, WS, NW, NS, EW.

Note that it is possible for more than one tile type $t \in T$ to share a set of input pads and therefore an address.

3.2.2. *Encoding of T.* We will now construct the string $w_\mathcal{T}$, which will represent the definition of T. Intuitively, $w_\mathcal{T}$ will be composed of a series of "entries." Each entry is associated to exactly one address of a tile type $t \in T$ and specifies the pads for the output sides of t. In this way, once the input sides for a supertile have formed, the corresponding pads can be used to form an address specifying (a set of) appropriate output pads. Note that since more than one tile type may share an address in a nondeterministic tile assembly system, more than one tile type may share a single entry.

We define an *entry* to be a string beginning with '#' followed by zero or more "sub-entries", each corresponding to a different tile type, separated by semicolons. Let A be the set of all binary strings representing every address created for each $t \in T$. The string $w_{\mathcal{T}}$ will consist of $1 + \max A$ entries for addresses 0 to $\max A$. The ith entry, denoted as e_i, corresponds to the ith address, which may or may not be in A (if it is not, then e_i is empty).

We say that a *sub-entry* consists of a string specifying the pads for the output sides of a tile type $t \in T$. Let e_i be the entry containing a given sub-entry (note that i is the address of e_i), and $T_i \subseteq T$ be the set of tile types addressable by i (i.e., the set of tile types for which i is a valid address). The entry e_i will be comprised of exactly $|T_i|$ sub-entries. For $0 \leq k < j$, the kth sub-entry in e_i, where $t_k \in T_i$ is the kth element of T_i (relative to some fixed ordering), is the string $\text{OUT}(N), \text{OUT}(E), \text{OUT}(S), \text{OUT}(W)$ (the commas in the previous string are literal) with $\text{OUT}(d) = \text{Bin}(\text{Pad}(t_k, d))^R$ if the glue for $\text{Pad}(t_k, d)$ is not g_{null} and d is not a component of the address i, otherwise $\text{OUT}(d) = \lambda$. Intuitively, a sub-entry is a comma-separated list of the (reversed) binary representations of the pads for an addressed tile type, but including only pads whose glues are not g_{null} and whose directions are not a part of the address (and therefore input sides). We will now use the string $w_{\mathcal{T}}$ to construct the lookup table $\mathbf{T}_{\mathcal{T}}$.

3.2.3. *Full specification of $\mathbf{T}_T$.* We now give the full specification for the lookup table $\mathbf{T}_{\mathcal{T}}$. First, define the following strings: $w_0 = \text{'>'}$, $w_1 = \text{'< \%\% >'}$, $w_2 = \text{'<'}$. Now let $\mathbf{T}_{\mathcal{T}}$ be as follows: $\mathbf{T}_{\mathcal{T}} = \text{sb}(w_0 \circ w_{\mathcal{T}} \circ w_1 \circ (w_{\mathcal{T}})^R \circ w_2)$, where, for strings x and y, $x \circ y$ is the concatenation of x and y, and $\text{sb} : \Sigma^* \to \Sigma^*$ is defined to "splice blanks" into its input: between every pair of adjacent symbols in the string x, a single '␣' (blank) symbol is inserted to create $\text{sb}(x)$. This splicing of blanks is required to be able to read from the table without "locking it from view", when reading the table for operations that require growing a column of tiles in towards the table (as opposed to away from it), a blank column is used, and for growing a column away from the table, a symbol column is used so that the symbol can be propagated to the top of the column for later copying.

3.2.4. *The Lookup Procedure.* In our construction, when a supertile t^* that is simulating a tile type $t \in T$ forms, we must overcome the following problem: once we combine the input pads (given as the output pads of the supertiles to which t^* attaches), how do we use $\mathbf{T}_{\mathcal{T}}$ to lookup the output pads for t^*? In what follows, we briefly describe how we achieve this. In other words, we show how an address, a string of random bits, and a copy of $\mathbf{T}_{\mathcal{T}}$ are used to compute the pad values for the non-input sides of a supertile. A detailed figure and example of this procedure can be found in the technical appendix.

For ease of discussion and without loss of generality, we assume that the row of tiles encoding $\mathbf{T}_{\mathcal{T}}$ (assembled West to East) and the column of tiles encoding an address and a random string of bits (assembled North to South at the West end of $\mathbf{T}_{\mathcal{T}}$) are fully assembled, forming an 'L' shape with no tiles in the area between them. For other orientations of the table and address the logical behavior is identical, simply rotated.

Intuitively, the assembly of the lookup procedure assembles column wise in a zig-zag fashion from left to the right. In the "first phase," a counter initialized to 0 is incremented in each column where the value of the tile in the representation of $\mathbf{T}_{\mathcal{T}}$ is a ';', thus counting up at each entry contained in $\mathbf{T}_{\mathcal{T}}$. Once that number matches the value of the given address (which, along with the random bits is copied through this procedure), the entry e

corresponding to that address has been reached and a new counter begins which counts the number of sub-entries n in that entry. Note that for directed tile systems, $n \leq 1$. Once the end of that entry is encountered, yet another counter, initialized to 0, begins and increments on each remaining entry until the end of the first copy of $w_{\mathcal{T}}$ is reached (the number n is propagated to the right). This counts the number of entries, denoted as m, between e and the end of the lookup table. The "second phase" is used to perform, in some sense, an operation equivalent to calculating $p = b \mod n$, where b is the binary value of the string of random bits required for the lookup procedure (this is how we simulate nondeterministic assemblies). This selects the index of the sub-entry in e which will be used, completing the random selection of one of the possibly many tile types contained in entry e.

In the current version of our construction, we merely use a random number selection procedure reminiscent of the more involved (but more uniform) random selection procedures discussed in [5]. Although it is possible to incorporate these more advanced techniques into our construction (and thus achieve a higher degree of uniformity in the simulation of randomized tile systems), we choose not to do so for the sake of simplicity.

Next, a reverse counter, a.k.a., a subtractor, counts down at each entry from m to 0, and by the way we constructed $\mathbf{T}_{\mathcal{T}}$, this final counter obtains the value 0 at the entry e (in the reverse of $w_{\mathcal{T}}$). Now, another subtractor counts from p to 0 to locate the correct sub-entry that was selected randomly. Finally, each pad in the sub-entry is rotated "up and to the right," and the group of pads is propagated through the remainder of the lookup table, thus ending with the values of the non-input pads represented in the rightmost column.

3.3. Supertile design

A supertile s is a subassembly in $\mathcal{U}_{\mathcal{T}}$ consisting of a $c \times c$ block of tiles from T, where c depends on the glue complexity of T. Each s can be mapped to a unique tile type $t \in T$. In our construction there are two logical supertile designs. The first, denoted *type-0*, simulates tile additions in $\mathcal{T}$ in which there are 2 input sides, each with glue strength $= 1$. The second, denoted *type-1*, simulates the addition of tiles via a single strength 2 bond.

While there are several differences in the designs of *type-0* and *type-1* supertiles, one commonality is how their edges are defined. Namely each input or output edge of any supertile is defined by the same sequence of variable values. Since the edges for each direction are rotations of each other, we will discuss only the layout of the south side of a supertile. From left to right, the tiles along the south edge of a supertile will represent a string formed by the concatenation (in order) of the strings: $\mathbf{T}_{\mathcal{T}}$, $\mathrm{Bin}(\mathrm{Pad}(t, S))$, $0^{c'}$, $\mathrm{Bin}(\mathrm{Pad}(t, S))$, and $\mathbf{T}_{\mathcal{T}}$. Note that c' is a constant that depends on the glue complexity of T.

3.3.1. *Type-0 Supertiles (i.e., simulating tiles that attach via two single-strength bonds).* When a tile binds to an assembly in $\mathcal{T}$ with two input sides whose glues are each single strength, there are $\binom{4}{2} = 6$ possible combinations of directions for those input sides: north and east (NE), north and south (NS), north and west (NW), east and south (ES), east and west (EW), and south and west (SW). These combinations can be divided into two categories, those in which the sides are opposite each other (NS and EW), and those in which the sides are adjacent to each other (NE, NW, ES, and SW).

Opposite Input Sides: Supertiles which represent tile additions with two opposite input sides, NS and EW, are logically identical to rotations of each other, so here we will only describe the details of a supertile with NS input sides. Figure 1 shows a detailed image
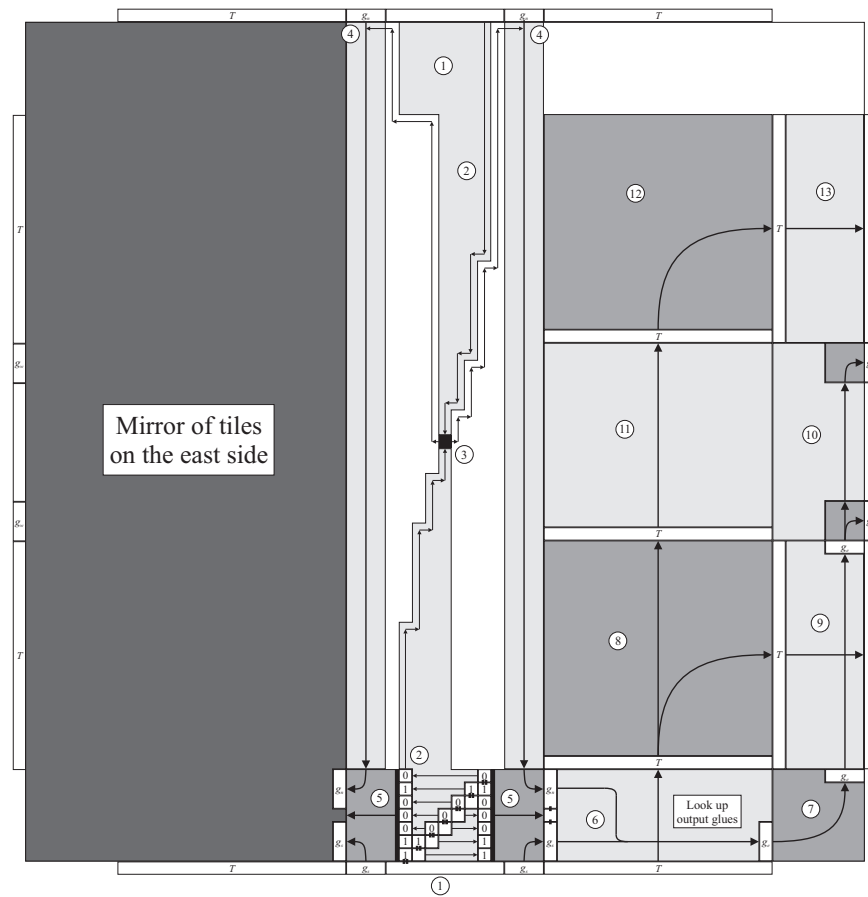
Figure 1:   NS supertile

depicting the formation of an NS supertile, with arrows giving the direction of growth for each portion and numbers specifying the order of growth. For ease of discussion and without loss of generality, we assume that the rows of tiles which form the input sides of a supertile have fully formed before any other part of the supertile assembles. The first portions to assemble are the center blocks to the interior of each input side, labeled 1. This subassembly forms a square in which a series of nondeterministic selections of tile types is used to generate a random sequence of bits. These bits are propagated to the left and right sides of the block, to ensure that each side uses the same random bits for the randomized selection after the sides have been "sealed off" from each other by "probes" described next. Once that block has completed, a log-width binary subtractor, which is half the width of the block, assembles. The subtractors from the north and south count down from a specified value (that depends on $\mathcal{T}$ and is encoded into the seed supertile) to 0, and shrink in width until they terminate at positions adjacent to the center square of the block. These subtractors are "probes" that grow to the center where the direction of the input sides (the *type*) is detected. It is at this point that the central (black in the figure) tile can attach. It is this tile which determines the *type* of the supertile (NS in this case) because it is unique to the combination of directions from which the inputs came. At this
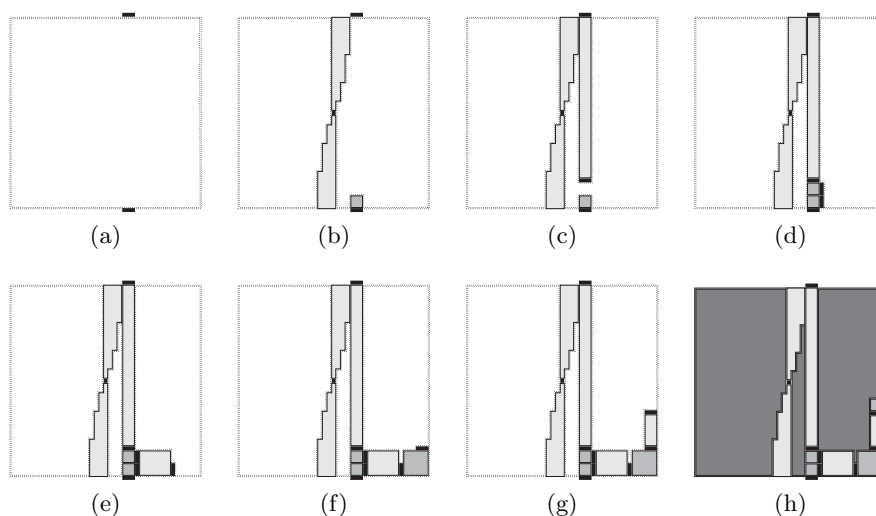
Figure 2: Intuitive depiction of (a portion of) the self-assembly of a type-0 supertile. Note that the lookup procedure is performed in (d) and (e).

point, symmetry is broken and two paths of tiles assemble from the center back towards the north side. They in turn initiate the growth of subassemblies which propagate the value of the north input pad down towards the South of the supertile. Once that growth nears the southern side, the two input pads are rotated and brought together, with this combination of input pads forming an *address* in the lookup table. In the manner described previously, this address along with the random bits generated within block 1 (which are also passed through block 5) is used to form the subassembly of block 6 whose southern row contains a representation of $\mathbf{T}_{\mathcal{T}}$ and results in the correct output pads being represented in the final column of that block. Note that Figure 1 only shows the details of the east side of the block since the West side is an identical but rotated version. Finally, subassemblies 7 through 13 form which rotate and pass the necessary information to the locations where it must be correctly deposited to form the output sides of the supertile. Every side of a supertile that is not an input side receives an output pad, even if it is for the null glue (in which case it does not initiate the growth of the input side of a possible adjacent supertile).

References

1. Leonard Adleman, Qi Cheng, Ashish Goel, and Ming-Deh Huang, *Running time and program size for self-assembled squares*, STOC '01: Proceedings of the thirty-third annual ACM Symposium on Theory of Computing (New York, NY, USA), ACM, 2001, pp. 740–748.
2. J. Albert and K. Čulik II, *A simple universal cellular automaton and its one-way and totalistic version*, Complex Systems **1** (1987), no. 1, 1–16.
3. Ho-Lin Chen and Ashish Goel, *Error free self-assembly with error prone tiles*, Proceedings of the 10th International Meeting on DNA Based Computers, 2004.
4. Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T. Schweller, and Diane L. Souvaine, *Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues*, Natural Computing **7** (2008), no. 3, 347–370.

5. David Doty, Jack H. Lutz, Matthew J. Patitz, Scott M. Summers, and Damien Woods, *Random number selection in self-assembly*, Proceedings of The Eighth International Conference on Unconventional Computation (Porta Delgada (Azores), Portugal, September 7-11, 2009), 2009.

6. David Doty, Matthew J. Patitz, and Scott M. Summers, *Limitations of self-assembly at temperature 1*, Proceedings of The Fifteenth International Meeting on DNA Computing and Molecular Programming (Fayetteville, Arkansas, USA, June 8-11, 2009), 2009, to appear.

7. B. Durand and Zs. Róka, *The game of life: universality revisited*, Tech. Report 98-01, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, January 1998.

8. Kenichi Fujibayashi, David Yu Zhang, Erik Winfree, and Satoshi Murata, *Error suppression mechanisms for dna tile self-assembly and their simulation*, Natural Computing: an international journal **8** (2009), no. 3, 589–612.

9. Grégory Lafitte and Michael Weiss, *Universal tilings*, STACS (Wolfgang Thomas and Pascal Weil, eds.), Lecture Notes in Computer Science, vol. 4393, Springer, 2007, pp. 367–380.

10. ———, *Simulations between tilings*, Tech. report, University of Athens, 2008.

11. ———, *An almost totally universal tile set*, TAMC (Jianer Chen and S. Barry Cooper, eds.), Lecture Notes in Computer Science, vol. 5532, Springer, 2009, pp. 271–280.

12. 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.

13. Urmi Majumder, Thomas H LaBean, and John H Reif, *Activatable tiles for compact error-resilient directional assembly*, 13th International Meeting on DNA Computing (DNA 13), Memphis, Tennessee, June 4-8, 2007., 2007.

14. Maurice Margenstern, *Cellular automata in hyperbolic spaces, vol. 1: Theory*, Old City Publishing, Philadelphia, 2007.

15. Nicolas Ollinger, *The intrinsic universality problem of one-dimensional cellular automata*, 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS) (H. Alt and M. Habib, eds.), LNCS, vol. 2607, Springer, 2003, pp. 632–641.

16. ———, *Intrinsically universal cellular automata*, Proceedings International Workshop on The Complexity of Simple Programs, Cork, Ireland, 6-7th December 2008 (T. Neary, D. Woods, A.K. Seda, and N. Murphy, eds.), EPTCS, vol. 1, 2009, arXiv:0906.3213v1 [cs.CC], pp. 199–204.

17. Mojżesz Presburger, *Ub̈er die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen*, welchem die Addition als einzige Operation hervortritt. Compte Rendus du I. Congrks des Mathematiciens des pays Slavs, Warsaw, 1930, pp. 92–101.

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

19. 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.

20. Paul W.K. Rothemund, Nick Papadakis, and Erik Winfree, *Algorithmic self-assembly of DNA Sierpinski triangles*, PLoS Biology **2** (2004), no. 12, 2041–2053.

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

22. David Soloveichik and Erik Winfree, *Complexity of compact proofreading for self-assembled patterns*, The eleventh International Meeting on DNA Computing, 2005.

23. ———, *Complexity of self-assembled shapes*, SIAM Journal on Computing **36** (2007), no. 6, 1544–1569.

24. Thomas LaBean Urmi Majumder, Sudheer Sahu and John H. Reif, *Design and simulation of self-repairing DNA lattices*, DNA Computing: DNA12, Lecture Notes in Computer Science, vol. 4287, Springer-Verlag, 2006.

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

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