Strict Self-Assembly of Discrete Self-Similar Fractals in the abstract Tile Assembly Model*

Florent Becker † Daniel Hader‡ Matthew J. Patitz§

Abstract

This paper answers a long-standing open question in tile-assembly theory, namely that it is possible to strictly assemble discrete self-similar fractals (DSSFs) in the abstract Tile-Assembly Model (aTAM). We prove this in 2 separate ways, each taking advantage of a novel set of tools. One of our constructions shows that specializing the notion of a *quine*, a program which prints its own output, to the language of tile-assembly naturally induces a fractal structure. The other construction introduces *self-describing circuits* as a means to abstractly represent the information flow through a tile-assembly construction and shows that such circuits may be constructed for a relative of the Sierpinski carpet, and indeed many other DSSFs, through a process of fixed-point iteration. This later result, or more specifically the machinery used in its construction, further enable us to provide a polynomial time procedure for deciding whether any given subset of $\mathbb{Z}^2$ will generate an aTAM producible DSSF. To this end, we also introduce the *Tree Pump Theorem*, a result analogous to the important *Window Movie Lemma*, but with requirements on the set of productions rather than on the self-assembling system itself.

*The full version of the paper can be accessed at `https://arxiv.org/abs/2406.19595`

†Laboratoire d'Informatique Fondamentale d'Orléans (UR4022), Université d'Orléans, `florent.becker@univ-orleans.fr`
Reasearch made possible by a CRCT research leave from the Conseil National des Universités, supported in part by ANR project Complexité des Systèmes Dynamiques Simples JCJC 2019 [19-CE48-0007-01]

‡Electrical Engineering and Computer Science Department, University of Arkansas, `dhader@uark.edu` Supported in part by National Science Foundation Grant FET-2329908.

§Electrical Engineering and Computer Science Department, University of Arkansas, `patitz@uark.edu` Supported in part by National Science Foundation Grant FET-2329908.

1 Introduction

In tile-based self-assembly or *tile-assembly*, primitive units called *tiles* combine via local interactions to form complex structures. The study of tile-assembly is ostensibly largely motivated by work in DNA-nanotechnology where synthetic DNA strands are made to combine, through the mechanism of nucleotide base pairing dynamics, into structures that resemble microscopic square tiles with selectively sticky sides. With careful design, these DNA tiles, under well chosen conditions (temperature, concentration, salinity, etc.), then tend to combine with each other to form desired structures or even follow the execution of algorithms [50, 44, 39, 42, 41, 13, 43, 2]. The abstract Tile-Assembly Model (aTAM) is a simplified, mathematical model of tile-assembly, and while the aTAM is often used to aid in the design of DNA-based self-assembling systems [41, 2, 13], it has been known since its introduction that the aTAM exhibits rich computational dynamics and even Turing universality[47]. Since then, theoretical investigations into the aTAM (and variants thereof) have unveiled a complex tapestry of hierarchies, asymptotic bounds, and intricate algorithmic constructions which has blossomed into a sort of computability and complexity theory for these uniquely geometric computational models [45, 1, 40, 6, 4, 21, 5, 14, 22, 30, 31, 32, 49, 23]. Even compared to other geometric computational models such as Cellular Automata [17, 33] or Tilings [38, 7, 46], the aTAM has uniquely asynchronous dynamics, making this landscape of results quite a change of scenery.

One method for investigating the aTAM involves characterizing the shapes and patterns it is capable of assembling. Early results in this direction found, for instance, that any finite shape may be constructed with an asymptotically optimal number of unique tile types (proportional to the Kolmogorov complexity of the shape) if one is willing to scale the shape by some finite but typically unfortunately large factor[45]. While the simulation of a Turing machine within the aTAM is relatively simple, used there (and in many other results) as a primitive *tile gadget* within a more complex aTAM construction, this result in particular illustrates an important specificity of the aTAM compared to more conventional models of computation[1], namely the relevance of how the computation is embedded into the geometric construction process of shapes. This makes it pertinent to consider what shapes are difficult (or impossible) to self-assemble. As is the case in many areas of study focused on geometry, fractals provide an interesting challenge.

While no single definition suffices for the term "fractal", convenient for our purposes are *discrete self-similar fractals* (DSSFs) [26], the limit points of self-substitution tilings of finite connected subsets of $\mathbb{Z}^2$. DSSFs are infinite and aperiodic which is enough in its own right to make a construction difficult, but further still DSSFs are sparse meaning that there is generally little contiguous space for tiles to simulate a Turing machine computation or something analogous. It's important to note here that this difficulty is only present in the context of *strict* self-assembly where tiles are only allowed to occupy locations in the target shape. This is in contrast to *weak* self-assembly wherein tiles may encroach into negative space, and the target shape is characterized by a specific subset of tile types. While it is trivial to weakly self-assemble a host of DSSFs, strict self-assembly of DSSFs has remained an open question. In [26] it was shown that the DSSF analog of the Sierpiński triangle is impossible to strictly assemble, and such impossibility results have since grown to include subsets of DSSFs such as so-called *pinch point* fractals, etc. [37, 15, 24, 3]. These results depend fundamentally on the fact that each stage of such a fractal is connected by a bounded number of tiles to the previous stage. Interestingly, this seems to hold "just barely", and in [28] and [26] it was shown that by allotting only a logarithmically increasing amount of additional connectivity between fractal stages, using processes called *lacing* and *fibering* respectively, it becomes possible to strictly self-assemble these DSSFs. Another feature of fractal shapes is a notion of dimension which may generally fall between integer values. More precisely, this notion generally assigns some value to how the amount of space occupied by the shape scales with its radius. In the context of DSSFs in the aTAM, ζ-dimension [11] serves as a convenient and common choice for this metric. It may be tempting to presume that a non-integer ζ-dimension plays a role in characterizing the assemble-able shapes. Yet, there have been aTAM constructions which can be configured to assemble shapes with any ζ-dimension among a dense subset of the interval $(1, 2)$, though the resulting shapes are distinctly not DSSFs nor are they even sparse[19].

In this paper, we close the question of strictly assembling DSSFs in the positive. We show not only that DSSFs can be assembled in the aTAM, we do so in 2 separate ways, and provide a polynomial time algorithm for deciding whether a DSSF generated by a given substitution rule can be assembled. While these results answer a long-standing open question in the field, more important are the novel and, we believe, delightful techniques

[1] with perhaps the exception of cellular automata

developed for their construction. One of our DSSF constructions arises naturally from the question "what does a quine look like in the aTAM?" where a quine typically refers to a program which prints its own description. Here we substitute typical notions of *description* in the context of a universal Turing machine with that of an *intrinsically universal* (IU) tileset in the aTAM [10]. An IU tileset is one capable of simulating all other aTAM systems in some class, not just symbolically as with a Turing machine, but geometrically so that the full dynamics of tile attachment are captured at scale. Our quine therefore is a tile system which grows into a scaled version of itself. Our IU tileset, while restricted to simulating just a subset (albeit useful subset) of all possible aTAM systems, is also certainly the smallest IU tileset published that has been fully implemented in an aTAM simulator and can be downloaded [20]. This construction can also be made to have essentially any ζ-dimension desired between 1 and 2. Our other DSSF construction relies on a novel scheme for identifying self-assemble-able shapes by abstracting the flow of information through an aTAM system's tiles into what we call *self-describing circuits*. These circuits translate naturally into the tile types necessary for assembly while allowing for precise reasoning about their functionality, and through fixed-point iteration we construct a self-describing circuit for a scaled version of a close relative of the Sierpiński Carpet. This latter construction generalizes to all generators (subsets of $\mathbb{Z}^2$ to be self-substitution-tiled) with sufficient *bandwidth*, a measure of a shape's ability to transmit information between their opposite sides. Namely, this generalization relies on having at least 2 disjoint paths between the both the north and south sides and the east and west sides of the generator simultaneously.

Finally, and perhaps most importantly, we characterize the set of generators that result in DSSFs which can be self-assembled in the aTAM. Specifically, we show that the assembly of a DSSF in the aTAM fundamentally relies on the bandwidth requirement of our circuit construction: its bandwidth requirement is actually a characterization up to a finite amount of iterations of the generator. Moreover, it may be determined for any generator shape in polynomial time using a maximum flow algorithm. This result is not only satisfyingly intuitive, especially given that the impossibility proof for constructing pinch point fractals depends on the same intuition, but additionally our proof involves a wide range of novel devices which we believe are of general interest to the tile-assembly community. This includes, among other things, the use of ordinal indexed assembly sequences to reason about infinite ones and an important theorem we call the *Tree Pump Theorem* which serves a similar purpose to the incredibly useful *Window Movie Lemma*[29]. Especially striking is the fact that the Tree Pump Theorem's hypotheses are about the shape of the productions of a system rather than the system itself, allowing anyone to prove in one fell swoop that a given shape cannot be obtained by *any* aTAM system.

2 Overview of the Concepts

Please see Figure 7 for the articulation of the concepts and results of the paper.

2.1 The abstract Tile Assembly Model The definitions we use for the aTAM are borrowed and adapted from [22] and we note that [40] and [26] are good introductions to the model for unfamiliar readers. The basic components of the aTAM are unrotatable 2-dimensional square *tiles*. Each side of a tile can have a *glue*, each of which is composed of a string *label* and an integer *strength*. All glues with strength 0 are assumed to have identical labels and are called *null* glues. A *tile type* is uniquely defined by an assignment of glues to its four sides, and a tile is an instance of a tile type. When two tiles are placed so that they have abutting sides, if those two sides share the same glue, they *bind* with the strength of that shared glue. An *assembly* is a configuration of tiles in $\mathbb{Z}^2$, and it is said to be τ-stable if bonds whose strengths sum to at least τ must be broken in order to separate the assembly into two or more components. We say that assembly α is a *subassembly* of β (written $\alpha \sqsubseteq \beta$) if dom $\alpha \subseteq$ dom β and α and β have the same tile types on their shared domain.

A *tile-assembly system* (TAS) is a triple $\mathcal{T} = (T, \sigma, \tau)$, where T is a tileset, σ is a finite τ-stable assembly called the *seed assembly*, and $\tau \in \mathbb{Z}^+$ is called the *binding threshold* (a.k.a. *temperature*). A tileset must be finite, but an infinite number of copies of each tile type are assumed to be present. If an assembly α can be produced in $\mathcal{T}$ by a series of single-tile τ-stable additions, starting from the seed assembly σ, we say that α is a *producible* assembly, and use $\mathcal{A}[\mathcal{T}]$ to denote the set of producible assemblies. If no additional tiles can τ-stably bind to α, we say that α is a *terminal* assembly, and use $\mathcal{A}_\square[\mathcal{T}]$ to denote the set of terminal assemblies. If $|\mathcal{A}_\square[\mathcal{T}]| = 1$, i.e., there is exactly one terminal assembly, we say that $\mathcal{T}$ is *directed*. Note that both of our DSSF constructions are directed. We say $\mathcal{T}$ *strictly self-assembles shape* S if for any $\alpha \in \mathcal{A}_\square[\mathcal{T}]$, dom $\alpha = S$.

When a tile initially attaches to an assembly, we call the sides with glues that perform that initial bonding its *input sides*, and any other sides with non-null glues, to which later attaching tiles bind, its *output sides*. We define

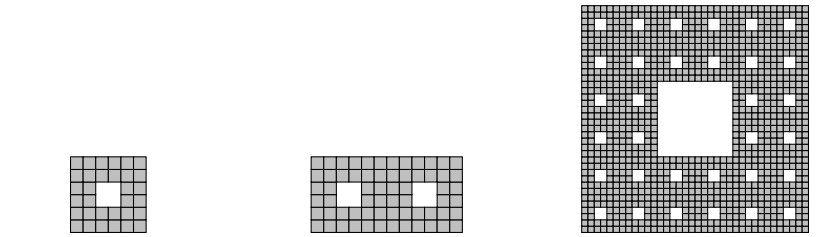


Figure 1: The substitution κ of the Sierpinski's Cacarpet.

standard aTAM systems as those which (1) are directed, (2) have for each tile type fixed and distinct sets of input sides and output sides, (3) produce no assemblies with an input side glue on its exterior, (4) have for each tile type the fact that its input sides are either a single side with a strength-2 glue or two diagonally adjacent sides with glues of strength-1, (5) have no glue mismatches in the single terminal assembly, and (6) never produce assemblies containing locations with cumulative inputs of strength 2 without a corresponding tile to attach in that location. Since the aTAM is computationally universal [47], and relatively straightforward algorithms exist for performing computational simulations of aTAM systems, there exists an aTAM tileset S such that, for any arbitrary aTAM system $\mathcal{T}$, a system using S that is appropriately seeded can computationally simulate $\mathcal{T}$. However, in [10] it was shown that a more natural notion geometry-preserving simulation is also possible. This notion of simulation, called *intrinsic simulation*, occurs when, for some $c \in \mathbb{Z}^+$, each $c \times c$ block of tiles of a simulating system $\mathcal{S}$ are mapped to a single tile of the system $\mathcal{T}$ being simulated. These $c \times c$ blocks of tiles are referred to as *macrotiles* and they must be formed in orderings that match the orderings of tile attachments in $\mathcal{T}$.

2.2 Discrete Self-Similar Fractal Shapes Fractals are usually defined in some continuous space such as $\mathbb{R}^2$ or $\mathbb{C}^2$. In the context of self-assembly, one needs to work with *discrete* fractals which are defined as fixed points of some 2 dimensional substitution. A substitution $\sigma_G(X) : \mathcal{P}(\mathbb{Z}^2) \to \mathcal{P}(\mathbb{Z}^2)$ based on a finite pattern G, called the *generator*, replaces each pixel in a pattern X with a copy of G.

Definition 2.1. (Dicrete Self-Similar Fractal) *Let G be a finite subset of $\mathbb{N}^2$ with $(0,0) \in G$. The discrete self-similar fractal shape with generator G is $G^\infty = \bigcup_{i \in \mathbb{N}} \sigma_G^i(\{(0,0)\}) \subset \mathbb{N}^2$. The i-th step of the fractal is $G^i = \sigma_G^i(\{(0,0)\}$.*

The *Sierpinski's Cacarpet* [2] is the self-similar fractal shape K^∞ with generator $K = \{0, \dots, 5\}^2 \setminus \{2,3\}^2$, represented on Figure 1; $\kappa = \sigma_K$ is the associated substitution.

3 Overview of our Results

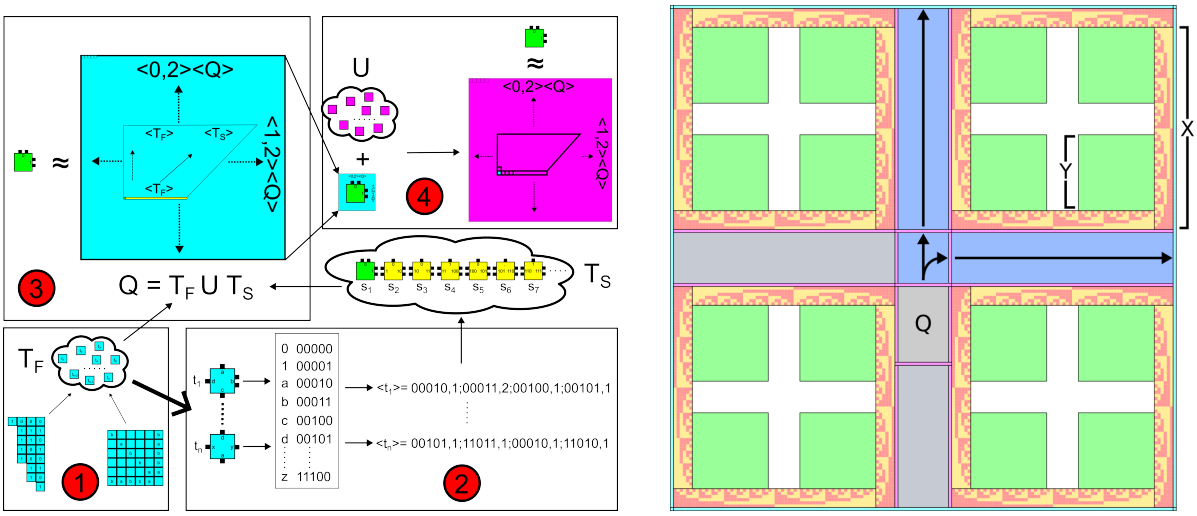
3.1 Fine Characterizations of the dynamics of the aTAM Three general results about the aTAM form the foundation of our fractal results. Each of them advances the field of self-assembly and has general relevance within it.

Standard aTAM systems are intrinsically universal and admit Quines

Theorem 3.1. *The class of standard aTAM tile assembly systems is intrinsically universal.*

We fully implemented the tile set using a relatively small number of tile types ($< 10,000$). In this section we describe the construction of an aTAM quine. In the broader context of computability theory, a quine refers to a program that, with no input, will output precisely its own description. Here, *description* is generally defined with respect to a universal machine U. If simulating U on the pair (d, x) yields the same results as machine $M(x)$ for all x, then d is a description of M in the context of U. A quine is then simply a description d of a machine $M_d(x) = U(d, x)$ which outputs d for all x. Here we describe a natural sense in which this definition can be

[2]Compared to Sierpinski's regular Carpet, the Cacarpet has a 2×2 hole, 2 layers of points around that hole and two occurences of "Ca" in its name.



(a) (b)

Figure 2: (a) A schematic depiction of the construction of the quine system. (1) Subsets of tile types capable of various algorithmic functions (e.g. binary counting, rotation of values, etc.) make up the "functional" tileset T_F. (2) The glue labels and tile types of T_F are encoded to generate the "seed row" tileset T_S that self-assembles a row presenting an encoding of T_F (plus some additional necessary information) via its northern glues. The seed tile is shown as green and the rest of the seed row as yellow. (3) Using the full tileset $Q = T_F \cup T_S$, the system grows from the seed tile to form the seed row, then the tiles of T_F cause upward growth that computes the definitions of the seed row tiles (via their northern glues and locations), appending those definitions to those provided by the seed row. This results in an assembly with the full definition of Q encoded in its northern glues. Further growth by the tiles of T_F causes that assembly to grow into a terminal square macrotile that is consistent with the definition of a seed assembly representing the quine system's seed tile for the IU tileset U. (4) If the terminal assembly from (3) were used as the seed assembly for a system including the tile types from U, that system would simulate the quine system. That is, each tile of Q would be simulated by a macrotile composed of tiles from U, resulting in a terminal assembly that is a macro-macrotile representing the seed tile of Q. (b) Schematic depiction of the formation of a macrotile square by the quine construction adapted for DSSFs. The dimension X, counted by binary counters, can be set to an arbitrary value as long as it is greater than the height of the grey rectangle (created by the earlier stages of the quine assembly growth), and the dimension Y can be set to be any value between 0 and $X/2$. Careful setting of X and Y values can allow for arbitrary percentages of the area contained within the macrotile to be empty space.

sufficiently specialized using the aTAM as the model of computation and intrinsic universality in place of Turing universality. In this context, the universal machine is replaced by an intrinsically universal tile set U. The notion of *description* here has to be modified to accommodate this change; for a description of a tile system $\mathcal{T}$ to be meaningful in the context of U requires that $\mathcal{T}$ be encoded in some way that tiles in U can use to intrinsically simulate $\mathcal{T}$. To this end, we introduce a notion of *seeding* where a tile system $\mathcal{S}$ is said to *seed* a system $\mathcal{T}$ with respect to U if $\mathcal{S}$ grows into the shape of a macrotile which can initiate an intrinsic simulation of $\mathcal{T}$ using the tiles in U.

DEFINITION 3.1. *Fix a tileset U which is IU for some class $\mathfrak{C}$ of TASs and let the corresponding representation and seed generation functions be $\mathcal{R}$ and S respectively. We say that a TAS $\mathcal{Q} = (Q, \sigma_Q, \tau_Q)$* seeds *the TAS $\mathcal{T} = (T, \sigma_T, \tau_T)$ with respect to $(U, \mathcal{R}, S)$ if: (1) $\mathcal{T} \in \mathfrak{C}$, (2) $|\sigma_Q| = 1$, (3) $\mathcal{Q}$ is directed, and (4) the macrotile seed assembly $S(\sigma_T)$ corresponding to σ_T has the same shape and glues presented on its exterior as the terminal assembly of $\mathcal{Q}$.*

In other words, for $\mathcal{Q}$ to seed $\mathcal{T}$ with respect to U means that $\mathcal{Q}$ will grow into an assembly which behaves in all ways as a valid macrotile seed for the intrinsic simulation of $\mathcal{T}$ by tiles in U.

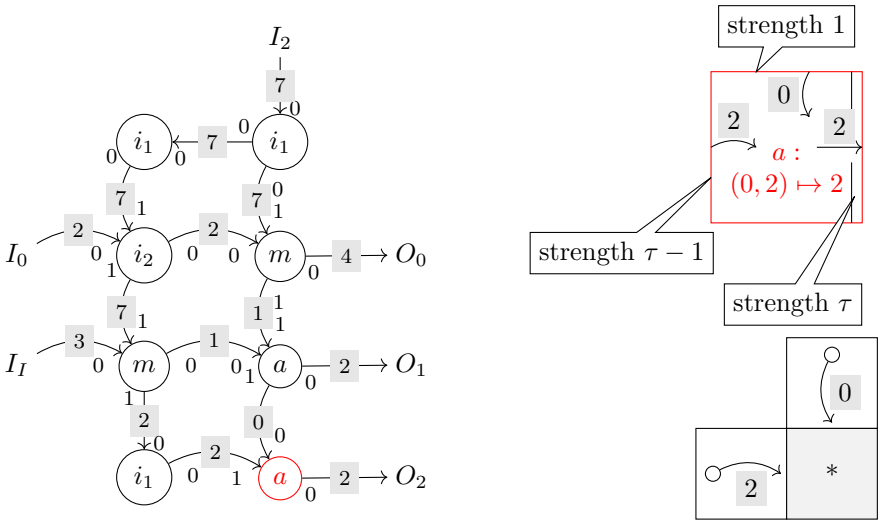


Figure 3: Left, a 2-digit by 1-digit multiplier computing 32 (left, written bottom-to-top) times 7 (top) is 224 (right, bottom-to-top). When a gate has two input wires, each of them bends towards the other wire. Right, the tile type corresponding to the a gate at the bottom-right of the circuit, and a neighborhood where it can attach. The temperature τ is the maximum in-degree of a gate in the circuit (here, 2).

DEFINITION 3.2. *Given IU tileset U (and its corresponding representation and seed generating functions $\mathcal{R}$ and S), an* aTAM quine *is a TAS $\mathcal{Q} = (Q, \sigma, \tau)$ that seeds itself with respect to $(U, \mathcal{R}, S)$.*

THEOREM 3.2. *There exists an aTAM quine.*

We prove Theorem 3.2 by construction, demonstrating an aTAM quine $\mathcal{Q} = (Q, \sigma, 2)$ for the tileset U that is IU for standard aTAM systems (and its corresponding representation and seed generating functions R and S) given in the proof of Theorem 3.1. A high-level, schematic depiction of $\mathcal{Q}$ can be seen in Figure 2a. Essentially, the tileset Q of $\mathcal{Q}$ consists of a subset of "functional" tile types, T_F, and a subset of "seed row" tile types, T_S. The seed row tile types grow into a row that has an encoding of the definitions of the tile types in T_F, and then the tiles of T_F attach to that row and grow into an assembly that computes the definitions of the tiles of T_S, so that the resulting assembly presents a full definition of Q. It then grows into a square macrotile that represents its own seed tile, having the glues of that tile represented on its sides along with the definition of Q.

Embedded Circuits and self-description *Self-describing circuits* are a novel device for presenting aTAM systems. These are generally useful for rigorous yet readable proofs of the behaviour of aTAM systems, which are crucial in the context of fractals.

A circuit is made of gates positioned on a subset D of $\mathbb{Z}^2$, as in Figure 3. Each of these gates has a *function*, computing the outputs of the gate from the inputs. The gate is embedded on a unit square according to its *wiring*, which states where its inputs come from and where its outputs go. The wiring and the function must be compatible: the wiring must have as many inputs as the function. A circuit is *evaluable* if it has no input wires, loops or infinite backwards paths. One can then evaluate the circuit, that is, give a value to each of its wires according to the functions of the gates.

An evaluable circuit is self-descriptive when there is a decoding function dec-gate which determines the gate at any position p of the circuit from the inputs it receives. If this is the case, the circuit can be self-assembled in the aTAM. In particular, the set $D \subset \mathbb{Z}^2$ of positions where gates appear can be strictly self-assembled in the aTAM.

THEOREM 3.3. *Let C be a self-describing evaluable circuit in which exactly one gate has no inputs. Then there is a standard aTAM system S_C which strictly self-assembles* $\mathrm{dom}(C)$.

The second column of Figure 3 shows the principle of the aTAM simulating such a circuit: each tile corresponds to a gate with a set of input/output values, and the set of glues is the set of wires with values on them. Each wire additionally encodes the set of directions with incoming wires in the gate, so the strength of each glue can be chosen so that each tile attaches when all its inputs are present. Each of the glues on the input sides of the gate then determines the set of input sides and by self-description, the combination of inputs on that set of sides is unique. Hence, the aTAM is directed and its production is exactly the circuit.

The Tree Pump Theorem This pumping principle in the style of [29] deals with skinny productions, that is those which do not encircle any square larger than m for some fixed m. The crucial property which makes them simple(-ish) is their bounded treewidth.

THEOREM 3.4. (TREE PUMP) *For any aTAM system $\mathcal{T} = (S, \sigma, \tau)$ with σ finite and connected, define the following sets of assemblies:*

- *for any integer m, $C_m[\mathcal{T}]$ is the set of assemblies of $\mathcal{T}$ which encircle an $m \times m$ square;*
- *for any real k and vector $\vec{d}$, $B_{k,\vec{d}}[\mathcal{T}]$ is the set of assemblies of $\mathcal{T}$ which do not cover any position $\vec{p}$ such that $\vec{p} \cdot \vec{d} > k + |\text{dom } \sigma|$*
- *for any vector $\vec{d}$, $P_{\vec{d}}[\mathcal{T}]$ is the set of* ultimately periodic *assemblies A of $\mathcal{T}$ such that there is a vector $\vec{p}$ with $\vec{p} \cdot \vec{d} > 0$ and a non-empty sub-assembly $a \sqsubseteq A$ such that $a + \vec{p} \sqsubseteq a$.*

Then, there is a function $F : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that for aTAM system $\mathcal{T}$ with n tile types and a 1-tile seed, integer m and unit vector $\vec{d}$ of $\mathbb{R}^2$,

$$\mathcal{A}_{\square}[\mathcal{T}] \cap (C_m[\mathcal{T}] \cup B_{F(n,m),\vec{d}}[\mathcal{T}] \cup P_{\vec{d}}[\mathcal{T}]) \neq \varnothing.$$

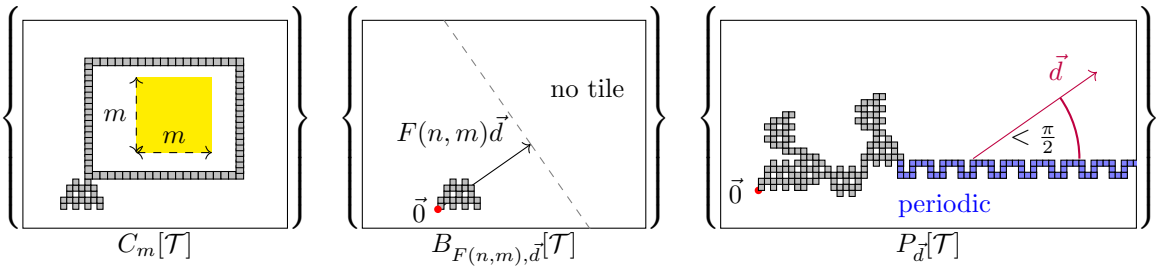


Figure 4: The three sets defined by theorem 3.4: $C_m[\mathcal{T}]$ is the assemblies which encircle an $m \times m$ square, $B_{F(n,m),\vec{d}}[\mathcal{T}]$ is the set of assemblies which do not reach further than $F(n,m)$ in direction $\vec{d}$, and $P_{\vec{d}}[\mathcal{T}]$ is the assemblies which contain a periodic path with period $\vec{p}$ such that $\vec{p} \cdot \vec{d} > 0$.

The cases of theorem 3.4 are detailed in Figure 4. In order to do a meaningful amount of computation, a self-assembling system with n tiles needs to encircle large squares —say, of size m, thus hitting $C_m[\mathcal{T}]$. If it does not, then its productions look very much like trees drawn on $\mathbb{Z}^2$ with a m-cell wide brush. A branch of that tree then behaves like a finite automaton. If that automaton stops, the assembly goes no further than $F(n,m)$ in any given direction $\vec{d}$, which gives a final production in $B_{F(n,m),\vec{d}}[\mathcal{T}]$. If not, these long branches must have an ultimately periodic behavior which gives a final production in $P_{\vec{d}}[\mathcal{T}]$. The proof proceeds in two parts, all along assuming that there are no $m \times m$ squares encircled by a production of $\mathcal{T}$: first the tree-like nature of the productions of $\mathcal{T}$ is formalized through a *tree decomposition*. Then, the proof considers generalized assembly sequences, with ordinal time and the freedom to escape $\mathbb{Z}^2$. In this generalized context, repeatedly using an adaptation of the Window Movie Lemma [29] on a long enough branch of the tree decomposition yields an ultimately periodic path. The full proof is given in Section 8, as it involves a fair few ancillary definitions.

3.2 From an aTAM Quine to a self-assembled DSSF Our first construction capable of strictly self-assembling DSSFs in the aTAM uses a smaller number of tile types (combining the tileset that is IU for standard aTAM systems and the quine tileset), but a very large scale factor.

THEOREM 3.5. *For $z, \epsilon \in \mathbb{Q}$, where $1 < z < 2$ and $\epsilon < 1$, there exists an aTAM system $\mathcal{F}_{z,\epsilon}$ such that $\mathcal{F}_{z,\epsilon}$ strictly self-assembles a DSSF whose ζ-dimension is in the range $(z-\epsilon, z+\epsilon)$ and whose generator has side length $O(2^{d/\epsilon})$.*

We prove Theorem 3.5 by construction. Given an arbitrary $z, \epsilon \in \mathbb{Q}$, where $1 < z < 2$ and $\epsilon < 1$, we demonstrate an aTAM system $\mathcal{F}_{z,\epsilon} = (T, \sigma, 2)$ that strictly self-assembles a DSSF whose ζ-dimension is in $(z - \epsilon, z + \epsilon)$. By including the IU tileset U in the quine construction, the macrotile resulting from the quine's growth will then enable tiles in U to simulate additional macrotiles. These macrotiles will grow in the same fashion as the individual tiles that attached to grow the quine in the first place. This process will continue at larger scales indefinitely. Because intrinsic simulation is a transitive relation (see Section 9) and because our IU system uses macrotiles with the same shape as our squared quine and does not place tiles in any macrotiles that will not map to tiles, this simulation at increasing scales produces a DSSF. Additionally, our construction enables the ζ-dimension to be tuned by adjusting the amount of empty space in each macrotile as depicted in Figure 2b

3.3 A self-assembled DSSF described as an Embedded Circuit The second construction relies on theorem 3.3: any shape which can be endowed with a self-describing circuit can be *strictly* self-assembled. Taking inspiration from [16] and others in this tradition, we observe that locally describing a recursive and self-similar structure such as that of a DSSF can be done by combining information about several levels in the substitution hierarchy in order to compute the address of each position. This affords strict-self-assembly of a DSSF with scale factor only 6, namely the Sierpiński Cacarpet K^∞ given on Figure 9. Alas, because the tileset is generated by a general theorem, its number of tile types ends up in the hundreds of millions.

THEOREM 3.6. *There is an evaluable circuit $C_\square$ with domain K^∞ and with an empty input bus which is self-describing. Moreover, $C_\square$ has only one gate without inputs.*

COROLLARY 3.1. *There is an aTAM system at $\tau = 2$ which self-assembles the DSSF pattern K^∞.*

In order to construct the circuit $C_\square$, three things are needed. First, its alphabet Σ, that is, the set of messages which are going to be circulating on its wires. Then the functions of its gates, and last the circuit itself, that is, the position of the gates and wires within K^∞. Because K^∞ is obtained as the fixed point of a substitution, $C^\square$ itself is going to be defined as the fixed point of a substitution κ. This substitution associates to each gate g a circuit, or macro-gate $\kappa(g)$ such that:

- each gate of $\kappa(g)$ announces its address within $\kappa(g)$ in its output values,
- each gate of $\kappa(g)$ announces a description of g in its output values, and
- as a circuit, $\kappa(g)$ simulates g.

In order for $C_\square$ to be well-defined, there is a special "seed" gate s at $(0, 0)$ which has no inputs and which appears in $\kappa(s)$ at position $(0, 0)$.

This is almost sufficient for self-description: in the fixed point of κ, every gate which has a predecessor within the same macro-gate receives the value of g as well as the position of a neighbor within $\kappa(g)$, from which it computes its own position. In each $\kappa(g)$ there is a gate p whose inputs all are from outside $\kappa(g)$; ensuring p only depends on where the inputs of g are, and that it does receive that information from its predecessor gives self-description.

The full details of the circuit are in Section 10; the idea is on Figure 5: each message in $C_\square$ contains two instances of pairs of a gate description and a position within K. The first one corresponds to the parent of the tile and the position of the tile within its parent, while the second one is used to simulate the parent.

3.4 A Characterization of Self-Assemblable DSSFs

Statement of the characterization The previous construction can be adapted to any discrete self-similar fractal within which the communication pattern used by $C_\square$ can be embedded. Whether a particular fractal is amenable to hosting such a communication pattern depends on the ability of its generator G to transport information to copies of itself around it, as captured by its cardinal *bandwidth*. The bandwidth $G[d \leftrightarrow d']$ between two directions $d, d' \in \{N, E, S, W\}$ is the number of disjoint paths through G from its d neighbor to its d' neighbor.

THEOREM 3.7. *There is a polynomial time algorithm which, on input G, decides whether:*

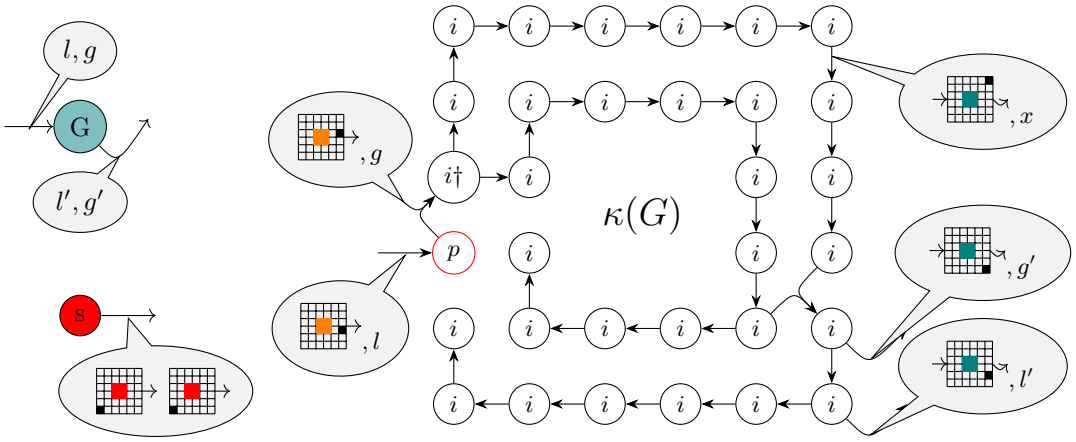


Figure 5: The principle of $C_\square$. Each gate G (top-left) is cut by κ into a circuit (right). The messages are composed of two parts. On the border of $\kappa(G)$, their second part corresponds to the messages of G. When a gate is cut, some information is lost: the type of its input wires. This information is reconstructed from the *output* wires of the predecessor, which are given in the local part of the messages from the neighboring $\kappa(G')$. The seed gate's message (bottom-left) declares "I am the tile at $(0,0)$ within a seed gate, at all scales".

1. *there is a k such that $G^k[N \leftrightarrow S] \geq 2$ and $G^k[E \leftrightarrow W] \geq 2$ and G^∞ can be strictly self-assembled in the aTAM, or*
2. *for all k, $G^k[N \leftrightarrow S] < 2$ and G^∞ cannot be strictly self-assembled in the aTAM, or*
3. *for all k, $G^k[E \leftrightarrow W] < 2$ and G^∞ cannot be strictly self-assembled in the aTAM.*

The fact that distinguishing between the cases can be done in polynomial time is quite plain. The interesting part is showing that this polynomial-time dichotomy is an actual characterization.

Bandwidth 2 is sufficient for strict self-assembly Let G, H be finite, connected shapes of $\mathbb{N}^2$, with $(0,0) \in G \cap H$. Then H is a *subconnector* of G, written $H \preceq G$ if G has a subgraph which is an edge subdivision of H, including the edges between adjacent copies of H when placing copies of G on a grid.

Now observe that whenever a graph G has K as a subconnector, the circuit of theorem 3.6 can be embedded into G^∞. Then changing the messages to carry addresses in G rather than in K yields a self-descriptive circuit with domain G^∞. It actually suffices that some finite iterate G^k admits K as a subconnector in order to conclude, since $(G^k)^\infty = G^\infty$.

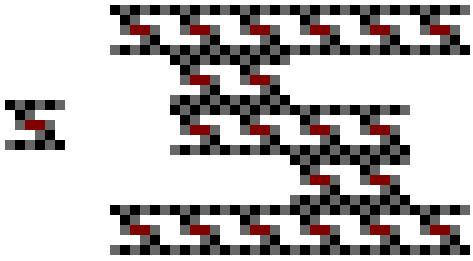


Figure 6: A generator where only the second iteration satisfies $G^2[N \leftrightarrow S] \geq 2$. Hence, $G[N \leftrightarrow S] = 1$, yet G^∞ can be strictly self-assembled.

LEMMA 3.1. *Let $G \ni (0,0)$ a finite, connected subgraph of $\mathbb{N}^2$ such that for some finite k, $K \preceq G^k$. Then there is a self-descriptive circuit with domain G^∞.*

When does $K \preceq G^k$? When $G[N \leftrightarrow S] \geq 2$ and $G[E \to W] \geq 2$: then $K \preceq G^3$ as the 2 disjoint paths in each direction turn into 8, which is enough to get K as a subconnector. This concludes the proof that condition 1 is sufficient to be able to strictly self-assemble G^∞.

Bandwidth 2 is necessary for strict self-assembly In cases 2 and 3 of theorem 3.7, strict self-assembly of G^∞ is impossible by a generalization of the impossibility result of Hendricks et al. [25].

THEOREM 3.8. *Let $G \ni (0,0)$ a finite shape of $\mathbb{N}$, and for $k \in \mathbb{N}$, $G^k = \sigma_G^k(\{(0,0)\})$.*

If $\sup_k(G^k[N \leftrightarrow S]) = 1$ or $\sup_k(G^k[E \leftrightarrow W]) = 1$, then G_∞ cannot be strictly self-assembled in the aTAM model unless $G^\infty = \mathbb{N}^2$.

Consider the case where for all k, $G^k[E \leftrightarrow W] = 1$. In G^∞, consider all edges which are East-West bridge of G^k for some k. Because $\mathcal{T}$ is finite, there is an infinite subset of those with the same glue on the bridge edge. From at least one of these glues g which appear infinitely often, it is possible to grow unboundedly far to the north. Let $\vec{p}$ be the vector joining two instances of g. By theorem 3.4, from g can grow either a periodic path going up to the north, or productions circling arbitrarily large squares. In both cases, this entails that all the G^k admit $\vec{p}$ as a period. Hence G^∞ is trivial.

Contents

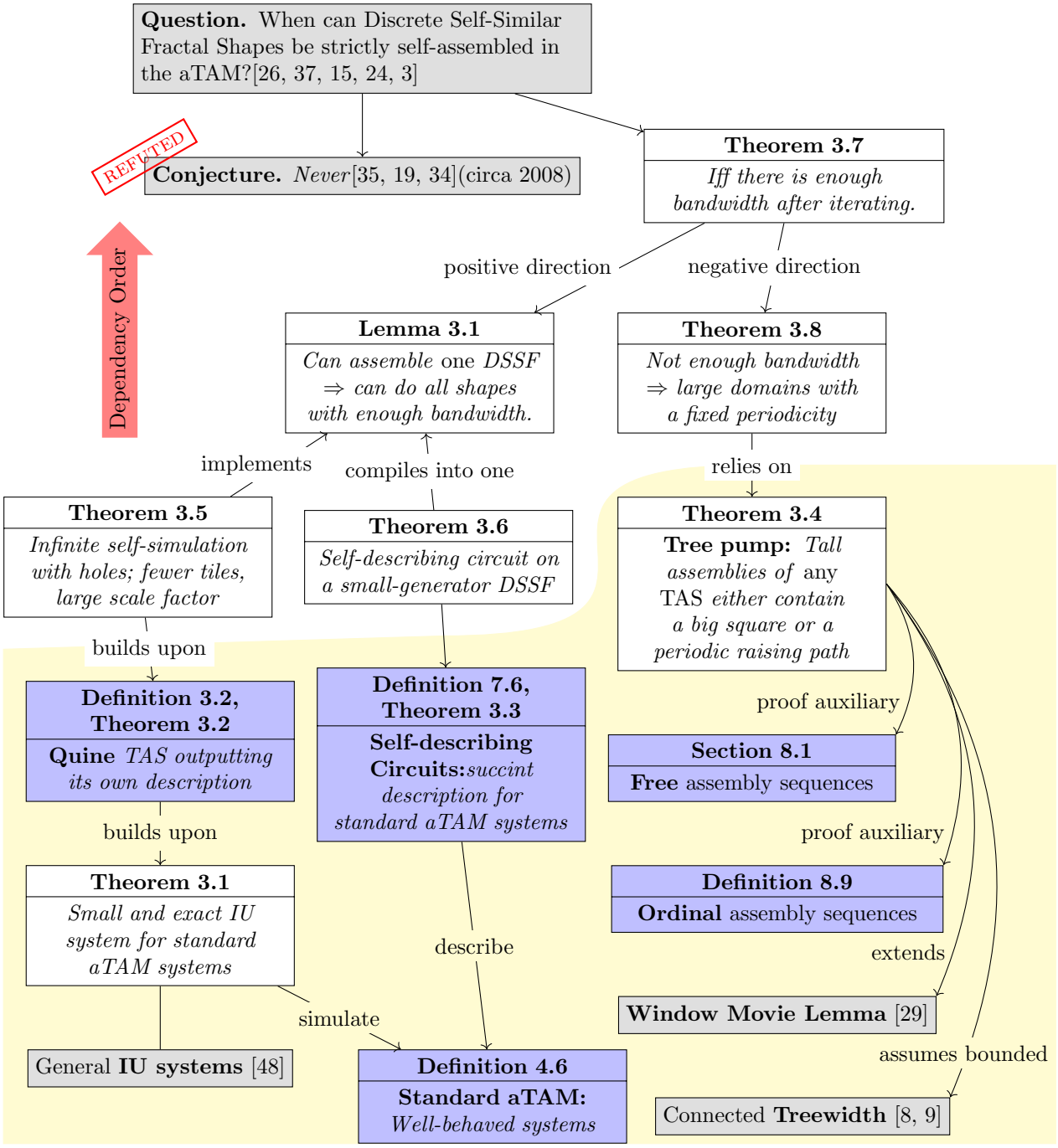


Figure 7: The articulation between the concepts and results of this paper. The gray boxes are from the literature, the blue boxes are new concepts. The results in the yellow part are of general interest outside the scope of fractals.

4 Definitions and Notations

In this section we provide mathematical definitions (extending the overview provided in Section 2) for the concepts and model used throughout the paper.

4.1 Generic Notations We define the four cardinal directions in $\mathbb{Z}^2$ as $D = \{N, E, S, W\}$ and identify them with the corresponding offsets from a location $\vec{l} \in \mathbb{Z}^2$: $O = \{(0,1), (1,0), (0,-1), (-1,0)\}$. A variable with an arrow such as $\vec{a}$ denotes a vector of values indexed by some set S. Given $s \in S$, a_s is the element of $\vec{a}$ associated with s. For any sets A, B and an indexing set S, given $\vec{a} \in A^S$ and $\vec{b} \in B^S$, the Cartesian product $\vec{a} \otimes \vec{b}$ is the vector $\vec{v} \in (A \times B)^S$ defined by $\forall s \in S, v_s = (a_s, b_s)$. Likewise, given $f : A^I \mapsto A^O$ and $g : B^I \mapsto B^O$, the function $f \otimes g : (A \times B)^I \mapsto (A \times B)^O$ is defined by $(f \otimes g)(\vec{a} \otimes \vec{b}) = f(\vec{a}) \otimes g(\vec{b})$.

The constructions in Section 7, Section 10 and Section 8 make heavy use of sets built as Cartesian products. In an attempt to make their exposition more readable and distinguish the function of each component of the product, they are given in a "record" or "object" like syntax using '{}' for record construction and "$x \cdot$ field" for field access. That is, given an integer k, k base sets $X_0, \ldots, X_{k-1}$ and a sequence of k labels such as $L = (\text{zeroth}, \text{first}, \text{second}, \ldots, \text{not-quite-k-th})$, the element $x = (x_0, x_1, x_2, \ldots, x_{k-1}) \in \Pi_i X_i$ is written $x = \{\text{zeroth} = x_0, \text{first} = x_1, \ldots, \text{not-quite-k-th} = x_{k-1}\}$. Symmetrically, x_0 can be extracted from x by writing $x{\cdot}\,\text{zeroth}$. A judicious set of labels can makes this notation actually useful in those sections.

4.2 Discrete Self-Similar Fractal Shapes In this paper, discrete self-similar fractal shapes are defined as fixed points of some 2 dimensional substitution. In [37], self-similar fractal shapes are defined as the union of an infinite hierarchy of shapes. The following definition is identical, as long as the generator contains $(0, 0)$.

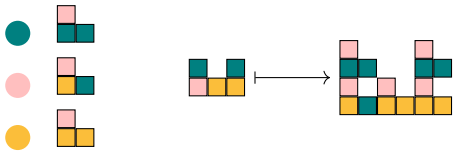


Figure 8: A colored substitution on a 3-colored alphabet $\Sigma = \{$●,●,●$\}$ is defined from a shape $G =$  and a coloring of G for each color of Σ. It can then be applied to any colored shape (right).

Explicitly defining a geometrical substitution will help, as that substitution can guide constructions happening on the fractal. If the generator does not contain $(0,0)$, the classical definition can be recovered by iterating the substitution associated with $G \cup (0,0)$ to reach the fix-point, then iterating once the substitution associated with G. The substitution σ_G based on a finite pattern G replaces each pixel in a pattern X with a copy of G.

Definition 4.1. (Rectangular substitution) *Let G be a finite subset of $\mathbb{N}^2$. Let $w = \max(\{x|(x,y) \in G\})$ and $h = \max(\{y|(x,y) \in G\})$. The substitution $\sigma_G : \mathcal{P}(\mathbb{N}^2) \to \mathcal{P}(\mathbb{N}^2)$ associated with G is defined by:*

$$\forall X \subset \mathbb{N}^2, \sigma_G(X) = \{p \in \mathbb{N}^2 | \lfloor p/G \rfloor \in X \wedge p \bmod G \in G\}$$

Using substitution allows manipulating the recursive definition of DSSF with richer data than one bit per position in the following way.

Definition 4.2. (Colored substitution) *Let X be an alphabet and G a finite subset of $\mathbb{N}^2$; given for each $x \in X$ a coloring $C(x) \in X^G$, the substitution σ_C associated with C is defined for each partial coloring $Y : \mathbb{N}^2 \dashrightarrow X$ by:*

$$\begin{cases} \sigma_C(Y) : \mathbb{N}^2 \dashrightarrow X \\ \sigma_C(Y) : \vec{z} \mapsto C(Y(\lfloor z/G \rfloor))(z \bmod G), \end{cases}$$

Where $\sigma_C(Y)(\vec{z})$ is defined whenever $Y(\lfloor z/G \rfloor)$ is defined and $z \bmod G \in G$.

Definition 4.3. (Self-Similar Fractal Shape) *Let G be a finite subset of $\mathbb{N}^2$ with $(0,0) \in G$. The discrete self-similar fractal shape with generator G is the following subset of $\mathbb{N}^2$:*

$$G^\infty = \bigcup_{i \in \mathbb{N}} \sigma_G^i(\{(0,0)\}).$$

The i-th step of the fractal is $G^i = \sigma_G^i(\{(0,0)\})$.

One self-similar fractal shape will be of particular interest to us in this paper, wich we name *Sierpinski's Cacarpet*. It is the self-similar fractal shape K^∞ with generator $K = \{0,\dots,5\}^2 \setminus \{2,3\}^2$, represented on figure 9. We note $\kappa = \sigma_K$ the associated substitution.

4.3 The Abstract Tile Assembly Model and Standard Systems A complete definition of the aTAM is given here. This section also introduces IO markings and the concept of a *standard TAS system*, which capture the common practices used in the literature to obtain intelligible aTAM systems.

Let Σ to be some alphabet with Σ^* its finite strings. A *glue* $g \in \Sigma^* \times \mathbb{N}$ consists of a finite string *label* and non-negative integer *strength*. There is a single glue of strength 0, referred to as the *null* glue. A *tile type* is a tuple $t \in (\Sigma^* \times \mathbb{N})^4$, thought of as a unit square with a glue on each side. A *tileset* is a finite set of tile types. We always assume a finite set of tile types, but allow an infinite number of copies of each tile type to occupy locations in the $\mathbb{Z}^2$ lattice, each called a *tile*. Given a tileset T, a *configuration* is an arrangement (possibly empty) of tiles in the lattice $\mathbb{Z}^2$, i.e. a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$. Two adjacent tiles in a configuration *interact*, or are *bound* or *attached*, if the glues on their abutting sides are equal (in both label and strength) and have positive strength. Each configuration α induces a *binding graph* B_α whose vertices are those points occupied by tiles, with an edge

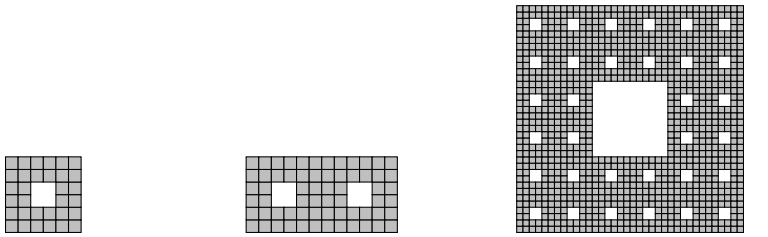


Figure 9: The substitution κ of the Sierpinski's Cacarpet.

of weight s between two vertices if the corresponding tiles interact with strength s. An *assembly* is a configuration whose domain (as a graph) is connected and non-empty. The domain dom $(\alpha) \subseteq \mathbb{Z}^2$ of an assembly α is known as its *shape*. We use $T^{\subset \mathbb{Z}^2}$ to denote the set of all assemblies of tiles in tile set T.

For some $\tau \in \mathbb{Z}^+$, an assembly α is τ-*stable* if every cut of B_α has weight at least τ, i.e. a τ-stable assembly cannot be split into two pieces without separating bound tiles whose shared glues have cumulative strength τ. Given a tile assembly system $\mathcal{T}$ and an assembly $\alpha \in \mathcal{T}^{\subset \mathbb{Z}^2}$, an *attachment candidate* is a pair $t@z$, with $t \in T$ and $z \in \mathbb{Z}^2$. It is *valid* if $z \notin$ dom α. It is *stable* if $\alpha' = \alpha \cup \{z \mapsto t\}$ is stable. If it is both valid and stable, it is an *attachment*, noted $\alpha \xrightarrow{t@z} \alpha'$. In that case, we say that α $\mathcal{T}$-*produces* α' *in one step*. That is, $\alpha \xrightarrow{t@z} \beta$ if β differs from α by the addition of a single tile of type t at position z. The $\mathcal{T}$-*frontier* is the set $\partial^{\mathcal{T}} \alpha = \left\{ z | \exists (t, \beta), \alpha \xrightarrow{t@z} \beta \right\}$ of locations in which a tile could τ-stably attach to α.

Given a TAS $\mathcal{T} = (T, \sigma, \tau)$, a sequence of $k \in \mathbb{Z}^+ \cup \{\infty\}$ assemblies $\alpha_0, \alpha_1, \ldots$ over $\mathcal{A}^T$ is called a $\mathcal{T}$-*assembly sequence* if, for all $i < k$, there is an attachment $t@z$ such that $\alpha_i \xrightarrow{t@z} \alpha_{i+1}$. The *result* of an assembly sequence is the unique limiting assembly of the sequence. For finite assembly sequences, this is the final assembly; whereas for infinite assembly sequences, this is the assembly consisting of all tiles from any assembly in the sequence. We say that α $\mathcal{T}$-*produces* β (denoted $\alpha \to^{\mathcal{T}} \beta$) if there is a $\mathcal{T}$-assembly sequence starting with α whose result is β. We say α is $\mathcal{T}$-*producible* if $\sigma \to^{\mathcal{T}} \alpha$ and write $\mathcal{A}[\mathcal{T}]$ to denote the set of $\mathcal{T}$-producible assemblies. We say α is $\mathcal{T}$-*terminal* if α is τ-stable and there exists no assembly that is $\mathcal{T}$-producible from α. We denote the set of $\mathcal{T}$-producible and $\mathcal{T}$-terminal assemblies by $\mathcal{A}_\square[\mathcal{T}]$. If $|\mathcal{A}_\square[\mathcal{T}]| = 1$, i.e., there is exactly one terminal assembly, we say that $\mathcal{T}$ is *directed*.

We say $\mathcal{T}$ *strictly self-assembles shape* P if for any $A \in \mathcal{A}_\square[\mathcal{T}]$, dom $A = P$. Weak assembly of a target shape is defined as follows. Given a TAS $\mathcal{T} = (T, \sigma, \tau)$, we allow each tile type to be assigned exactly one *color* from some set of colors C. Let $C_P \subseteq C$ be a subset of those colors, and $T_{C_P} \subseteq T$ be the subset of tiles of T whose colors are in C_P. Given an assembly $\alpha \in \mathcal{A}[\mathcal{T}]$, we use dom $_{C_p}(\alpha)$ to denote the set of all locations of tiles in α with colors in C_P.

Definition 4.4. (Input and Output sides) *Let $\mathcal{T} = (T, \sigma, \tau)$ be a TAS in the aTAM, and let $\alpha \in \mathcal{A}[\mathcal{T}]$ be a producible assembly in $\mathcal{T}$. If $\vec{t}$ is a tile of type $t \in T$ that attaches to α, we call the subset of sides on which $\vec{t}$ has non-null glues that form bonds with α at the time that $\vec{t}$ attaches its* input sides *and denote them as $\texttt{IN}(\vec{t}) \subseteq D$. Let $\texttt{G}(\vec{t}) \subseteq D$ be the set of sides of $\vec{t}$ containing non-null glues. We refer to the set $\texttt{G}(\vec{t}) \setminus \texttt{IN}(\vec{t})$ as the tile's* output sides*, and denote them as $\texttt{OUT}(\vec{t})$.*

Definition 4.5. (IO marked tile set) *Let $\mathcal{T} = (T, \sigma, \tau)$ be a TAS in the aTAM. We say that T is an* IO marked tile set *if there exists an ordered set of 4 unique symbols $\texttt{IO}_s = \{s_N, s_E, s_S, s_W\}$ such that the following conditions hold for every tile type $t \in T$ where a tile of type t is not contained in σ:*

1. *For some subset of sides of t whose glues are non-null such that the strengths of those glues sum to exactly τ, the glue labels of the glues on those sides end with the symbols from $\texttt{IO}_s$ corresponding to the directions of those sides. We say that those sides are* input marked*.*
2. *For all sides of t that contain non-null glues which are not input marked, the glue labels of the glues of glues on those sides ends with the symbols from $\texttt{IO}_s$ corresponding to the opposite directions of those sides. We say that those sides are* output marked*.*

For each $t \in T$ such that a tile of type t is contained in σ, it may have no sides that are input marked, or input marked sides whose glue strengths sum to $\leq \tau$.

LEMMA 4.1. *Let $\mathcal{T} = (T, \sigma, \tau)$ be an aTAM TAS where T is IO marked. If all exposed glues on the perimeter of σ are output marked, then for all $\alpha \in \mathcal{A}[\mathcal{T}]$, all exposed non-null glues on the perimeter of α are output marked.*

Proof. We prove Lemma 4.1 by induction. Our base case is σ which by definition is in $\mathcal{A}[\mathcal{T}]$ and is given to have all exposed non-null glues on its perimeter to be output marked. Our induction hypothesis is that, given producible assemblies $\alpha, \beta \in \mathcal{A}[\mathcal{T}]$ such that $\alpha \rightarrow_1^{\mathcal{T}} \beta$, if all non-null glues on the perimeter of α are output marked, then the same holds for β. This must be true because, since T is IO marked, whatever tile $\vec{t}$ of whichever type $t \in T$ attaches to α to form β must only have glues whose strengths sum to exactly τ, the minimum possible that allow it to bind, that are input marked. Since all glues on the perimeter of α are output marked, and the only way for glues on t and the perimeter of α to match and thus form bonds is for them to be on tile sides of opposite directions and one to be input marked and the other output marked (since those are the same symbols for the opposite sides), every input marked glue of $\vec{t}$ must bind during its attachment. This leaves only null and output marked glues remaining on $\vec{t}$ to possibly be exposed and added to the perimeter of β. Therefore, all exposed non-null glues on the perimeter of β must also be output marked and our induction hypothesis holds and Lemma 4.1 is proven. ∎

COROLLARY 4.1. *Given aTAM TAS $\mathcal{T} = (T, \sigma, \tau)$ where T is IO marked and all exposed glues on the perimeter of σ are output marked, then for every tile type $t \in T$, all tiles of type t that attach to any producible assemblies in $\mathcal{T}$ must have the exact same input sides and output sides (noting that output sides are not required to ever form bonds with other tiles).*

Corollary 4.1 follows immediately from Lemma 4.1 since the tiles of T are IO marked and therefore can only ever use (exactly) their input marked sides as their input sides when initially binding, and any remaining non-null glues are output marked and also available as output sides. Therefore, for such systems we will also use the notation $\texttt{IN}(t)$ and $\texttt{OUT}(t)$, referring to tile *type* t, in addition to $\texttt{IN}(\vec{t})$ and $\texttt{OUT}(\vec{t})$ referring to individual tiles of type t.

Throughout this paper, we will use the following two sets of symbols to IO mark tile sets: $\texttt{IO}_{s1} = \{\text{V}, <, \wedge, >\}$ and $\texttt{IO}_{s2} = \{\text{VV}, <<, \wedge\wedge, >>\}$. Intuitively, they can be thought of as "pointing into the tile" when on input sides, and out of the tile on output sides.

DEFINITION 4.6. (STANDARD TAS) *Let $\mathcal{T} = (T, \sigma, 2)$ be a TAS in the aTAM. We say that $\mathcal{T}$ is* standard *if and only if:*

1. *$\mathcal{T}$ is directed,*
2. *T is IO marked,*
3. *For every $t \in T$, the sides that have input markings are either exactly (1) a single glue of strength-2, or (2) two diagonally adjacent strength-1 glues (i.e. not on opposite sides),*
4. *All glues on the exterior of σ are output marked, and*
5. *There are no mismatches in the terminal assembly (i.e. all adjacent pairs of tile sides in $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$ have the same glue label and strength on both sides)*
6. *In any tile location between two diagonally adjacent non-null glues or next to a strength-2 glue, exactly 1 tile may attach.*

Note that throughout the literature of the aTAM, most constructions consist of standard aTAM systems or systems that could trivially be turned into standard systems by adding IO markings (e.g. [27, 36, 45, 40]).

4.4 Encodings of aTAM tile types and systems Here we provide definitions related to the ways in which aTAM tile types and systems can be encoded for use in systems that simulate their behaviors.

DEFINITION 4.7. (GLUE ENCODING) *Let G be a set of glue labels, Σ an alphabet, and $f_G : G \to \Sigma^*$ be an injective (i.e. one-to-one) function mapping glue labels to strings. For $g \in G$, we say that $f_G(g)$ is a* glue encoding of g over Σ. *That is, $f_G(g)$ is a unique representation of g among all glues in G using the fixed alphabet Σ.*

DEFINITION 4.8. (GLUE LOOKUP ENTRY) *Given an IO marked tile set T, some $t \in T$, and alphabets Σ and Σ' with $\Sigma \subseteq \Sigma'$, we define a* glue lookup entry of t over Σ' *as a string $s \in \Sigma'^*$ that consists of glue encodings of the input sides of t under Σ followed by glue encodings of the output sides of t under Σ, possibly separated and/or surrounded by additional characters in $\Sigma' - \Sigma$.*

DEFINITION 4.9. (GLUE LOOKUP TABLE) *Given an IO marked tile set T and alphabets Σ and Σ' with $\Sigma \subseteq \Sigma'$, we define a* glue lookup table of T over Σ' *as a string $s \in \Sigma'^*$ that consists of a glue lookup entry of each $t \in T$ over Σ, possibly separated and/or surrounded by additional characters in $\Sigma' - \Sigma$.*

4.5 Intrinsic Simulation in the aTAM This section describes what it means for a TAS to intrinsically simulate another TAS. Intuitively, intrinsic simulation of a system $\mathcal{T}$ by another system $\mathcal{S}$ is done with respect to some scale factor $c \in \mathbb{Z}^+$ such that $c \times c$ squares of tiles in $\mathcal{S}$, called *macrotiles*, represent individual tiles in $\mathcal{T}$, and there is a representation function that is able to map the macrotiles in $\mathcal{S}$ to tiles in $\mathcal{T}$ (or empty space) and thus interpret the assemblies of $\mathcal{S}$ as assemblies in $\mathcal{T}$. Furthermore, the progression of the mapped macrotiles from $\mathcal{S}$ faithfully mimics the addition of tiles in $\mathcal{T}$.

In the following definitions, it will be assumed that $\mathcal{T} = (T, \sigma_\mathcal{T}, \tau_\mathcal{T})$ is a TAS being simulated by another TAS $\mathcal{S} = (S, \sigma_\mathcal{S}, \tau_\mathcal{S})$. Furthermore, let $\mathbb{Z}_n$ be the set $\{0, 1, \ldots, n-1\}$ equipped with the typical notions of modular arithmetic. Given a positive integer c called the *scale factor*, a *c-block macrotile* over S is a partial function $\mu : \mathbb{Z}_c \dashrightarrow S$. That is, μ assigns tiles from S to some subset of the locations in a $c \times c$ block of locations. If the domain of μ is empty, then the macrotile is called *empty*. During a simulation, it is assumed that the lattice $\mathbb{Z}^2$ is divided regularly into c-block macrotiles so that the origin occupies the south-westernmost location in the corresponding macrotile block. Given a general assembly $\alpha \in \mathcal{A}^\mathcal{S}$ and some coordinates $(x, y) \in \mathbb{Z}^2$, we can recover the macrotiles from α by letting $\mathcal{M}^c_{x,y}[\alpha]$ refer to the c-block macrotile in α whose southwest corner occupies location (cx, cy). In other words $\mathcal{M}^c_{x,y}[\alpha](i, j) = \alpha(cx + i, cy + j)$, when defined, for $(i, j) \in \mathbb{Z}^2_c$.

A partial function $R : \mathscr{B}^S_c \dashrightarrow T$ is called a *macrotile representation function* from S to T if for any $\mu, \nu \in \mathscr{B}^S_c$ where $\mu \sqsubseteq \nu$ and $\mu \in \operatorname{dom} R$, then $R(\mu) = R(\nu)$. In other words, if R maps a macrotile μ to a tile type in T, then any additional tile attachments to μ should not change how it maps under R. This is clearly required since tiles in the aTAM cannot detach or change to other tile types so macrotiles should not be able to change their representation once assigned. With respect to an assembly sequence $\alpha_1, \alpha_2, \ldots$ in $\mathcal{S}$, a macrotile at location (x, y) is said to *resolve* into tile type $t \in T$ at step i if $\mathscr{M}^c_{x,y}[\alpha_{i-1}]$ is not in the domain of R but $R(\mathscr{M}^c_{x,y}[\alpha_i]) = t$.

Given a c-block macrotile representation function R, The corresponding *assembly representation function* is denoted $R^* : \mathscr{A}^S \to \mathscr{A}^T$ and is defined so that $R^*(\alpha') = \alpha$ exactly when $\alpha(x, y) = R(\mathscr{M}^c_{x,y}[\alpha'])$ for all $(x, y) \in \mathbb{Z}^2$. In other words, the assembly representation function is just the result of applying the macrotile representation function to each macrotile in $\mathcal{S}$. Additionally, the notation $R^{*-1}(\alpha)$ is used to refer to the producible preimage $\{\alpha' \in \mathcal{A}[\mathcal{S}] \mid R^*(\alpha') = \alpha\}$ of the assembly representation function R^* on α. In other words $R^{*-1}(\alpha)$ is the set of all $\mathcal{S}$-producible assemblies that map to α under R^*.

In order for intrinsic simulation to be a distinctly unique and meaningful definition compared to traditional notions of simulation, a restriction on R^* is necessary. Given an assembly $\alpha' \in \mathcal{A}[\mathcal{S}]$ let $\alpha = R^*(\alpha')$. It is then said that α' *maps cleanly* to α under R^* if all tiles in α' appear in the macrotile blocks which have either themselves resolved to tiles in α or which are adjacent (not diagonally) to macrotiles which have resolved. This means that tiles may not exist in macrotile blocks of α' mapping to locations of α that could not possibly admit a tile attachment due to their distance from all tiles in α. Tiles are allowed to attach in macrotile blocks of α' adjacent to resolved macrotiles to determine if the macrotile block might eventually resolve to a tile. These unresolved macrotiles adjacent to resolved macrotiles are called *fuzz* macrotiles since at scale the tiles attaching in them resemble fuzzy hairs on the surface of a resolved assembly.

DEFINITION 4.10. *Formally, α' maps cleanly to α' when for every non-empty block $\mathscr{M}^c_{x,y}[\alpha']$, there exists a vector $(u, v) \in \mathbb{Z}^2$ with length less than or equal to 1, such that $\mathscr{M}^c_{x+u,y+v}[\alpha'] \in$ dom R.*

DEFINITION 4.11. *We say that $\mathcal{S}$ and $\mathcal{T}$ have* equivalent productions *(under R), written $\mathcal{S} \Leftrightarrow_R \mathcal{T}$, if the following conditions hold:*

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

DEFINITION 4.12. *We say that $\mathcal{T}$* follows *$\mathcal{S}$ (under R), written $\mathcal{T} \dashv_R \mathcal{S}$, if for all $\alpha', \beta' \in \mathcal{A}[\mathcal{S}]$, $\alpha' \to^{\mathcal{S}} \beta'$ implies $R^*(\alpha') \to^{\mathcal{T}} R^*(\beta')$.*

In the following definition let ${R^*}^{-1}(\alpha)$ refer to the producible pre-image of the assembly representation function R^* on α. That is ${R^*}^{-1}(\alpha) = \{\alpha' \in \mathcal{A}[\mathcal{S}] | R^*(\alpha') = \alpha\}$, so that ${R^*}^{-1}(\alpha)$ consists of all $\mathcal{S}$-producible assemblies that represent α under R.

DEFINITION 4.13. *We say that $\mathcal{S}$* models *$\mathcal{T}$ (under R), written $\mathcal{S} \vDash_R \mathcal{T}$, if for every $\alpha \in \mathcal{A}[\mathcal{T}]$, there exists a non-empty subset $\Pi_\alpha \subseteq {R^*}^{-1}(\alpha)$, such that for all $\beta \in \mathcal{A}[\mathcal{T}]$ where $\alpha \to^{\mathcal{T}} \beta$, the following conditions are satisfied:*

1. *for every $\alpha' \in \Pi_\alpha$, there exists $\beta' \in {R^*}^{-1}(\beta)$ such that $\alpha' \to^{\mathcal{S}} \beta'$*
2. *for every $\alpha'' \in {R^*}^{-1}(\alpha)$ and $\beta'' \in {R^*}^{-1}(\beta)$ where $\alpha'' \to^{\mathcal{S}} \beta''$, there exists $\alpha' \in \Pi_\alpha$ such that $\alpha' \to^{\mathcal{S}} \alpha''$.*

For each $\alpha \in \mathcal{A}[\mathcal{T}]$, we call the corresponding set Π_α the stem set *of α.*

In Definition 4.13 above, the stem set Π_α of α is defined to be a set of assemblies representing α from which it is still possible to produce assemblies representing all possible β producible from α. Informally, the first condition specifies that all assemblies in Π_α can produce some assembly representing any β producible from α, while the second condition specifies that any assembly α'' representing α that may produce an assembly representing β is producible from an assembly in Π_α. In this way, Π_α represents a set of the earliest possible representations of α where no commitment has yet been made regarding the next simulated assembly. Requiring the existence of such a set Π_α for every producible α ensures that non-determinism is faithfully simulated. That is, the simulation cannot simply "decide in advance" which tile attachments will occur.

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

Let REPR denote the set of all supertile representation functions (i.e., c-block supertile representation functions for some $c \in \mathbb{Z}^+$). Define $\mathfrak{C}$ to be a class of tile assembly systems, and let U be a tile set. Note that each element of $\mathfrak{C}$, REPR, and $\mathcal{A}^U_{<\infty}$ is a finite object, hence encoding and decoding of simulated and simulating assemblies can be defined to be computable via standard models such as Turing machines and Boolean circuits.

DEFINITION 4.14. *We say a tile set U is* intrinsically universal *(IU) for $\mathfrak{C}$ if there are computable functions $\mathcal{R} : \mathfrak{C} \to \mathsf{REPR}$ and $S : \mathfrak{C} \to \mathcal{A}^U_{<\infty}$ such that, for each $\mathcal{T} \in \mathfrak{C}$, there is a constant $c \in \mathbb{Z}^+$ such that, letting $R = \mathcal{R}(\mathcal{T})$, $\sigma_\mathcal{T} = S(\mathcal{T})$, and $\mathcal{U}_\mathcal{T} = (U, \sigma_\mathcal{T}, \tau)$, $\mathcal{U}_\mathcal{T}$ simulates $\mathcal{T}$ at scale c and using supertile representation function R.*

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

5 Standard aTAM Systems are Intrinsically Universal

We now prove that the class of standard aTAM systems is intrinsically universal by providing a construction of a tileset that is IU for it. This universal tileset will be combined with the quine system presented in the next section to make systems that are capable of self-simulation.

5.1 Outline of IU construction Here we describe in broad strokes how our construction is designed. Technical details regarding individual tile gadgets are deferred to Section 5.2. It's important to emphasize here that there is no single best way to implement such a tileset; many of the choices we made are entirely arbitrary. This section therefore does not describe a fundamental construction, but rather one of infinitely many possible ways to implement a tileset IU for standard systems. It should also be noted that the restriction to standard systems makes designing an IU tileset significantly simpler than if the entire class of aTAM systems were to be simulated. This allows our IU tileset to consist of considerably fewer than 10,000 tile types while more general IU tilesets can easily contain orders of magnitude more [22].

In this section, it will be assumed that U is refer to the IU tileset and we will describe the various parts of our construction under the pretext that a standard system $\mathcal{T} = (T, \sigma_T, \tau_T \leq 2)$ is being intrinsically simulated by a standard system $\mathcal{U} = (U, \sigma_U, \tau_U = 2)$.

Definition 5.1. *We say that the intrinsic simulation of a system $\mathcal{T}$ by another $\mathcal{U}$ is* pristine *if no terminal assembly in $\mathcal{U}$ has any fuzz tiles. That is, any macrotile of $\mathcal{U}$ containing a tile maps to a tile type in $\mathcal{T}$ under the representation function.*

In addition to being universal for all standard systems, our IU tileset U also has the property that if the seed macrotile σ_U encodes all of the tiles in T according to the convention described in Section 5.2.1, then the simulation of $\mathcal{T}$ by $\mathcal{U}$ will be pristine. This property is essential for our DSSF construction since otherwise the resulting shape would contain tiles in what should be empty locations.

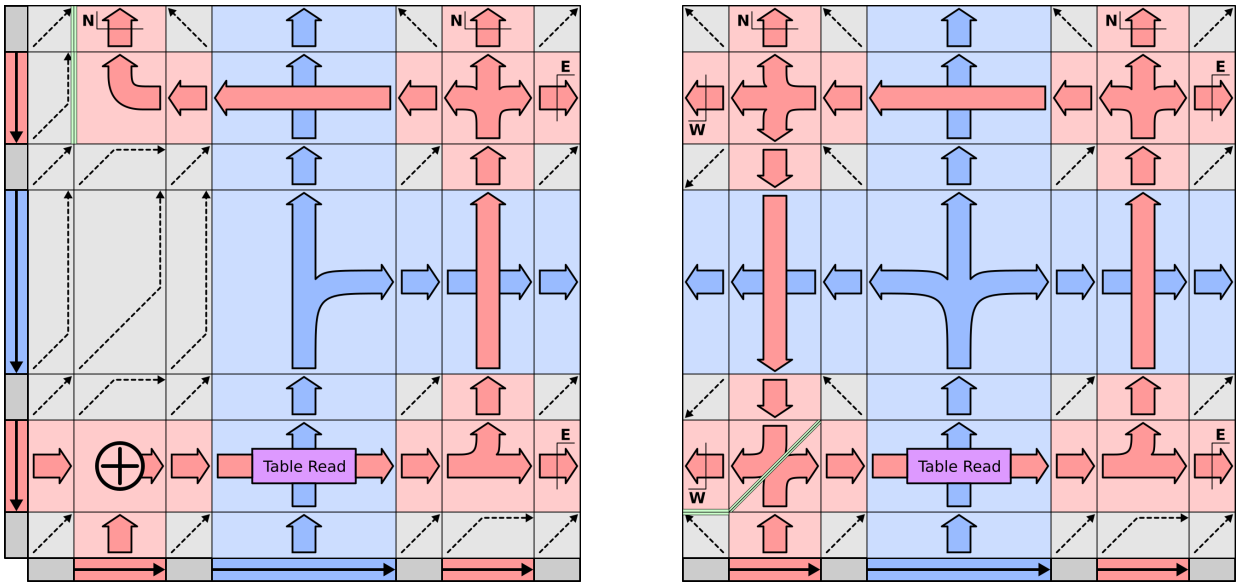
5.1.1 Macrotile structure In this construction, we insist that every tile attachment occurring within a macrotile block is dedicated solely to either determining how that macrotile block will resolve or to presenting information along the edges of that macrotile block to indicate to neighboring macrotiles how it has resolved. We also ensure that each macrotile block will only contain tiles in $\mathcal{U}$ if the corresponding tile location in $\mathcal{T}$ will contain a tile in the terminal assembly. This can be guaranteed since standard systems are locally deterministic.

The 4 sides of a completed macrotile in $\mathcal{U}$ each encode 2 pieces of information: the tileset T in the form of a data structure called a *glue table*, and the glue to be presented along the corresponding side of the T-tile type t being represented. Note that this encoding is predicated on the fact that t has a glue with positive strength on the corresponding side. If any of the sides of t contain the null glue, then the corresponding side of the macrotile will entirely consist of null glues.

Each side of a macrotile may be logically divided into 7 sections each separated by dedicated glues. The layout of these sections is symmetric along the side of the macrotile, though the data contained in these sections is decidedly not symmetric. The center section encodes the glue table for T. It is surrounded by two equally sized padding sections consisting only of strength-1 blank glues that encode no information. These are surrounded by identical copies of glue encodings called *glue signatures* whose purpose is to indicate which glue from $\mathcal{T}$ the macrotile side represents. Finally, these are surrounded by two additional, equally sized padding sections. Note that the outer and inner padding sections need not be the same size and only serve to accommodate space inside the macrotile for passing information around.

The boundaries that divide a macrotile side into 7 sections extend into the macrotile so that the entire block is divided into 49 logical sections as illustrated in Figure 10. Note that these sections may differ in their functionality depending on whether a cooperative or strength-2 attachment is being simulated in the current macrotile, though there are only a small number of distinct functional tasks that a section may perform. These tasks include the following:

1. **Filler sections** Filla region with generic tiles up to the boundaries (indicated by gray boxes with thin, dotted arrows),
2. **Propagation sections** propagate and/or rotate a sequence of symbols (encoded by the glues of tiles) from one side of the section to one or more other sides (indicated by thick colored arrows in Figure 10),
3. **Propagation intersection sections** pass two sequences of symbols through each other to the opposite sides of the section (indicated by thick arrows of different colors passing over each other),
4. **Glue combination sections** combine glue signature information from a pair of neighboring strength-1 glues (indicated by the symbol $\oplus$ in Figure 10),



(a) The layout of macrotile sections when simulating a cooperative attachment

(b) The layout of macrotile sections when simulating a strength-2 attachment

Figure 10: Blue arrows indicate the propagation of the glue table while red arrows indicate the propagation of glue signatures. Gray boxes indicate sections which are filled with generic tiles. Thin green lines indicate places where tiles growing from opposite directions place null glues between themselves so that mismatches are avoided.

5. **Glue table read sections** read from the glue table to convert an input glue signature into an output glue signature, and
6. **Glue output sections** "clean up" an output glue signature which may contain several output glue encodings (one for each output side) so that only the one for a specific output side is presented (indicated by a thick colored arrow struck-through by a thin line with a cardinal direction symbol).

5.1.2 Section boundary tiles Each section of a macrotile is separated by special *section boundary* tiles. These grow along the sides of adjacent macrotiles and completed sections within the current macrotile to drive the growth of each consecutive section. These tiles keep track of the current section within a macrotile using a local coordinate system of rows and columns. Each macrotile section exists between a pair of rows and a pair of columns. For instance, when simulating a cooperative attachment, the first tile to attach between two diagonally adjacent macrotiles is a section boundary tile representing the corner between row 0 and column 0. Additional section boundary tiles may then attach to form the entirety of row 0 and column 0 of the macrotile and the sections may begin to grow by attaching to these boundary tiles.

Which function a section of a macrotile performs is dependent on the local coordinates of the section within the macrotile, so the section boundary tiles enforce that the correct function is performed in each section. Additionally, the boundary tiles also serve to pass information between macrotile sections when necessary and keep track of whether the current macrotile represents a strength-2 attachment or a cooperative one.

5.1.3 Initiating the growth of macrotiles In standard systems, there are 2 distinct ways tiles may attach: using a single strength-2 glue or by the cooperation of two strength-1 glues from diagonally adjacent tiles. Growth of a macrotile simulating a cooperative attachment is initiated by a cooperative attachment between the diagonally adjacent macrotiles, while growth of a macrotile simulating a strength-2 attachment is initiated by a single strength-2 glue which is guaranteed to exist by convention on the counter-clockwise-most glue signature on the face of a completed macrotile. In this latter case, the strength-2 glue always exists in the position of the most-significant bit of the binary encoding of the glue from $\mathcal{T}$, though this convention represents a completely arbitrary choice.

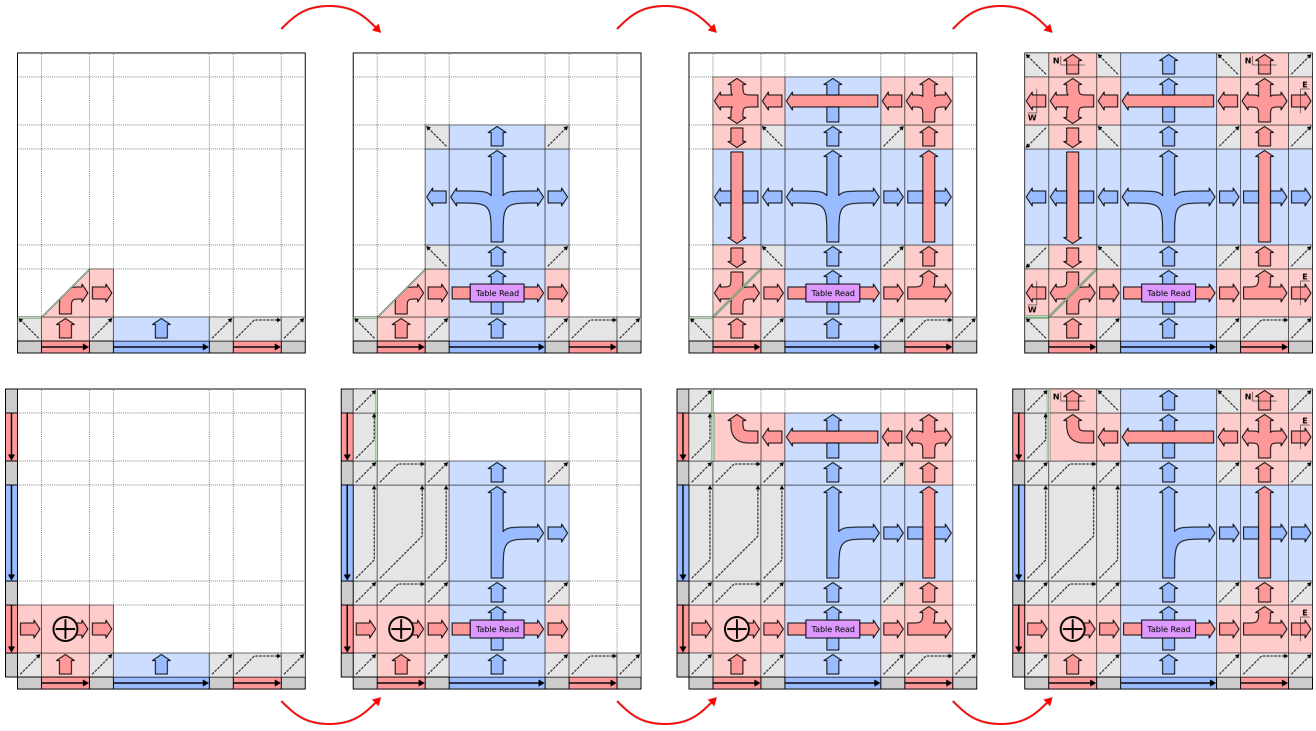


Figure 11: Different phases of IU macrotile growth when simulating strength-2 attachment to an existing macrotile on the south (top) and cooperative attachment to existing macrotiles from the south and west (bottom). For strength-2 attachments, the encoding of the $\mathcal{T}$ glue (red) is simply rotated towards where the glue table (blue) read will take place. For cooperative attachments, the two glue encodings are combined into one before being propagated towards the glue table. Reading from the glue table is then performed while also propagating the glue table information towards the center of the macrotile. The resulting glue signature encoding all output glues is propagated around the center of the macrotile and finally presented along the relevant edges of the macrotile.

When simulating a cooperative attachment, the current macrotile receives input glue encodings from 2 adjacent macrotiles. These encodings are combined into a single glue signature representing the combined information from both glues. For macrotiles simulating a strength-2 attachment, this step is skipped since there is only 1 input glue for the simulated tile. Instead, the glue signature representing this single input glue is simply rotated so that it is propagated towards the glue table. In either case, our convention for glue signatures ensures that glue signatures representing a single glue are the same width as signatures representing pairs of glues and that the direction of the glues is indicated implicitly in the encoding. See Section 5.2.1 for specific details on our glue signature conventions.

5.1.4 Reading from the glue table Whether it be from a pair of input glues or a single input glue, the next step of the macrotile growth process involves using the input glue signature to read from the glue table. The glue table encodes all of the tile types in T as pairs of input and output glue signatures. Reading from the glue table is done in a specific section that receives the input glue signature along one axis and the glue table encoding along the other. The input glue signature is propagated along the length of the glue table encoding using tiles that perform symbol matching logic to determine if the input glue signature exactly matches any of the input glue signatures in the glue table. If a match is found, the corresponding output glue signature is rotated in place of the input glue signature and additional matching is skipped. On the side opposite of the input glue signature is then the corresponding output glue signature, ready to be propagated to the output sides of the macrotile. Additionally, the encoding of the glue table is propagated towards the center of the macrotile unaltered so that it may be distributed to the output sides for use in adjacent macrotiles.

The representation function for our IU simulation may be defined by the output of the glue table reading gadget. Since the glue table contains information regarding all tiles to be simulated in a fixed order, the location where the glue table read gadget succeeds in finding a match uniquely corresponds to the tile type into which the macrotile should resolve. This macrotile resolution can occur as soon as the match is detected, but the exact tile attachment used to determine this is immaterial so long as resolving the macrotile occurs before any output information is presented to neighboring macrotiles.

Our convention for initiating macrotile growth ensures that the only time any tiles may attach inside of a macrotile is when two diagonally adjacent macrotiles present non-zero-strength glues to common neighbor or when a macrotile presents a strength-2 glue to a neighbor. Consequently, assuming the glue table contains encodings of all tiles to be simulated, by our requirements on standard systems, our IU simulation of standard systems will never contain fuzz tiles in unresolved macrotiles in any terminal assembly. In other words, because of the way macrotiles are initiated, assuming all tiles to be simulated are encoded in the glue table and the system to be simulated is standard, the simulation will be pristine.

5.1.5 Propagating output information The definition of standard systems ensures that the glue table will always have an entry corresponding to any input glue signature that it may see. After reading from the glue table, all that remains is to propagate the resulting glue signature (which represents the output glues of the tile from $\mathcal{T}$ represented by the macrotile) to the output sides of the macrotile. This is mostly done using standard tile gadgets for rotating and propagating sequences of symbols, though care needs to be taken to ensure that each side of the macrotile only has the part of the glue signature dedicated to that side. For instance, an output glue signature might contain information for glues to be presented on the north and east faces of a macrotile, but the north face should only present the north part of the signature and the east face the east part. This is handled by the *glue output sections* which, using two passes over the signature, removes any part of the signature corresponding to other directions and replaces the most significant bit of the counter-clockwise-most signature on a face with a strength-2 glue in the case that the encoded glue is itself strength-2.

5.2 Technical details

5.2.1 Conventions and Encodings Throughout this construction, we choose and stick with a few conventions for encoding glues and tilesets. These conventions are essentially completely arbitrary and the only motivation for our choice was ease of implementation. There are certainly other choices that could easily work, and ours is in no way canonical.

Glue Signatures For glues $g_N, g_E, g_S, g_W \in \mathscr{G}[T]$, let the notation $\langle g_N, g_E, g_S, g_W \rangle$ be called a *glue signature* and denote a mapping from the cardinal directions (north, east, south, and west in that order) to the respective glues. In any situation where a glue in a glue signature is the strength-0 glue, the symbol 0 can be used in the notation. For instance, the glue signature $\langle 0, g_1, g_2, 0 \rangle$ would assign the glue g_1 to the direction "east", the glue g_2 to "south", and the strength-0 glue to both the north and west directions. Keep in mind that glue signatures are not required to refer to a tile type with the corresponding glues on its respective sides. Instead, they are used as abstract data types in the construction, simply for when it is more convenient to keep track, for whatever reason, of multiple glues from different directions rather than individual glues. A glue signature is nothing more than the information contained in assigning a glue from $\mathscr{G}[T]$ to each cardinal direction.

Now let $\phi : \mathscr{G}[T] \times \{N, E, S, W\} \rightarrowtail \{0,1\}^m$ be an injective mapping, called the *glue encoding function*, from glues of $\mathscr{G}[T]$ and the corresponding side of the tile on which the glue appears to binary strings of a fixed length m. Furthermore, it is assumed that there is only a single strength-0 glue in $\mathscr{G}[T]$ which is mapped under ϕ to the string 0^m. In the case that T contains tile types using multiple distinct strength-0 glues, it can easily be modified to use only one without altering its behavior by choosing one of the strength-0 glues arbitrarily and using that in place of the others. Additionally, it is assumed that all strength-1 glues map under ϕ to binary strings whose left (most significant) bit is 0 and all strength-2 glues map to binary strings whose left bit is 1. Since $\mathscr{G}[T]$ is finite, it is not difficult to devise an implementation of ϕ and length m which satisfy these constraints.

Given a glue signature $\langle g_1, g_2, g_3, g_4 \rangle$, the notation $\phi\langle g_1, g_2, g_3, g_4 \rangle$ is abused to refer to the concatenation of the binary strings $\phi(g_1)$, $\phi(g_2)$, $\phi(g_3)$, and $\phi(g_4)$, each prefixed by a special symbol "#". In other words, $\phi\langle g_1, g_2, g_3, g_4 \rangle$ is a string over the alphabet $\{0, 1, \#\}$ which matches the following regular expression.

$$\#(0|1)^m\#(0|1)^m\#(0|1)^m\#(0|1)^m$$

Since ϕ is injective, this assignment of strings to glue signatures is invertible so that a glue signature may be recovered from its assigned string representation.

Glue Tables During our intrinsically simulation of $\mathcal{T}$, the tiles of $\mathcal{U}$ will need to somehow encode all relevant details regarding the dynamics of $\mathcal{T}$. Particularly, this includes information about which tile types of $\mathcal{T}$ can attach along different parts of a growing assembly. To accomplish this, each macrotile in $\mathcal{U}$ keeps track of a data structure called the *glue table* which encodes the full tile set T as a mapping between glue signatures.

A glue table Γ is a finite sequence of *entries*, ordered pairs of glue signatures representing an input and output. Recall that glues in T are IO-marked. Each tile type t in T is assigned an entry in the glue table where its input glues are assigned to the input signature of the glue table, and its output glues are assigned to the output signature. Output glues are assigned directly to the output signature so that the direction of the glue in the signature is the direction of the face on which the glue appears. For instance, if an output glue of type g_o appears on the north face of tile type t, then the output signature corresponding to tile type t will have g_o as its north component. Input glues on the other hand, are assigned to the input glue signature with the opposite direction of the face on which they appear. For example, if an input glue g_i appears on the east face of t, then it will be assigned to the west component of the corresponding input signature. All other components of the input and output glue signatures are assigned 0. Figure 12 illustrates what these signatures would look like for 2 example tile types.

Given a tile set T of size n to be simulated, a glue table Γ may be constructed by iterating over each tile type $t \in T$. For each tile type, an entry is constructed using the glues of t as described above. Specifically, the input glues of each tile are assigned the signature s^t_{in} and the output glues to the signature s^t_{out}. A glue table therefore is a sequence of pairs of glue signatures:

$$\Gamma = (s^{t_1}_{\text{in}}, s^{t_1}_{\text{out}}), (s^{t_2}_{\text{in}}, s^{t_2}_{\text{out}}), \ldots, (s^{t_n}_{\text{in}}, s^{t_n}_{\text{out}})$$

The order of entries in the glue table is immaterial since this construction will parse glue tables using string matching logic rather than index counting logic. Given a glue table Γ and a glue signature encoding function ϕ, the notation $\phi\Gamma$ is abused to refer to the concatenation of glue signature encodings using the special separator symbols "\$", "—$_1$", and "—$_2$" in the following way. For each tile type t encoded in the glue table with input signature s^t_{in} and output signature s^t_{out}, the corresponding glue table entry will be encoded as the string

$$\phi s^t_{\text{in}} \; |_1 \; |_2 \; \phi s^t_{\text{out}}$$

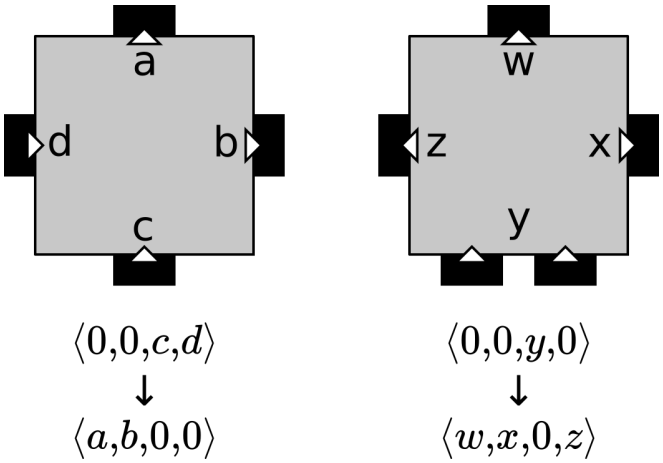


Figure 12: Two example tile types (top) and their corresponding glue table entry signatures (bottom). Glue IO-marks are indicated by small triangles, pointing into the tile type for input glues and out of the tile type for output glues. The left tile type has 2 strength-1 glues as inputs on its south and west sides. The right tile type has a single strength-2 glue as input on its south side.

In other words, the encoding of each entry is the encoding of the input and output glue signatures separated by "$|_1$" and "$|_2$". Each of these entries is then prefixed by the separating symbol "$" and concatenated together so that a glue table encoding may look like:

$$\$\ \phi s_{\text{in}}^{t_1}\ |_1\ |_2\ \phi s_{\text{out}}^{t_1} \$\ \phi s_{\text{in}}^{t_2}\ |_1\ |_2\ \phi s_{\text{out}}^{t_2} \cdots \$\ \phi s_{\text{in}}^{t_n}\ |_1\ |_2\ \phi s_{\text{out}}^{t_n}$$

This encoding is summarized by a grammar whose terminal characters are $\{0, 1, \#, \$, |_1, |_2\}$, non-terminals are T, E, and S representing the encodings for a glue table, glue table entry, and glue signatures respectively, and production rules are:

$$\begin{aligned}
T &\to \$ET \\
T &\to \$E \\
E &\to S|_1|_2S \\
S &\to \#(0|1)^m\#(0|1)^m\#(0|1)^m\#(0|1)^m
\end{aligned}$$

5.2.2 Common gadgets The bulk of our IU construction relies on the array of common gadgets we present here. When simulating a tile attachment, most sections within a macrotile will contain some variation of these gadgets. The gadgets presented here all exhibit rectilinear growth meaning that within each gadget, all tiles have the same pair of adjacent input directions. Rectilinear growth begins with the cooperative attachment of a tile in one corner of a section and ends with the attachment of a tile in the diagonally opposite corner.

Note also that it's possible to combine these gadgets so their functionality operates simultaneously in the same section. This enables, for instance, tiles to rotate a sequence of symbols in multiple directions at the same time. This does come at the cost of tile complexity however; combining gadgets requires tiles with glue labels that systematically concatenate the relevant information from the corresponding glue labels of the gadgets to be combined. The specific method of concatenating glue labels to form new ones is essentially a free choice, but effectively, this results in the tile complexity of the combined gadget being proportional to the product of tile complexity of each component gadget.

The first gadget, illustrated in Figure 13 propagates information from each of the input sides of a section to the opposite sides. This same gadget can be used to propagate information in just one direction by simply ensuring that the orthogonal direction propagates blank symbols that don't encode any information. The tile complexity for this gadget is proportional to the product of the number of distinct symbols that need to be propagated in both directions.

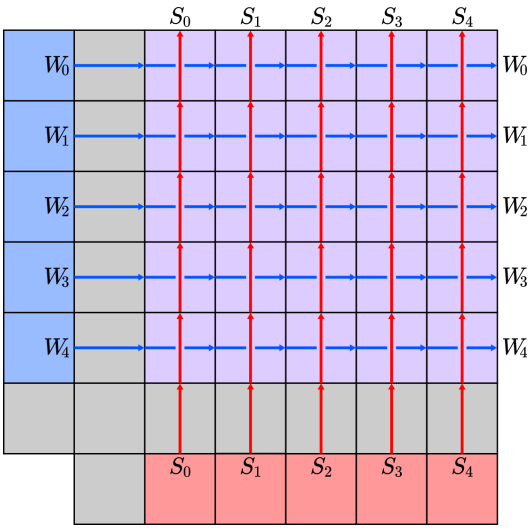


Figure 13: A schematic of tiles used to propagate two bit strings, one from south to north and the other from west to east. Arrows indicate the direction in which information is being propagated. Note that each tile in this illustration represents a schema of tile types. For instance, if the string of symbols being propagated contain only the binary bits 0 and 1, then there would need to be 4 distinct tiles to allow the information to propagate, one for each pair of bits (one from the west and one from the south).

The next gadgets, illustrated in Figure 14, rotate symbol information from one direction to an orthogonal one.

5.2.3 Combining glue signatures When simulating cooperative attachments between strength-1 glues, our IU tileset must combine the glue signatures presented from both of the cooperating macrotiles. Figure 15 depicts how information is propagated and combined as tiles attach.

5.2.4 Reading a glue table Throughout the IU construction, glue tables will be encoded by rows of tiles along the sides of each macrotile in the simulation. The purpose of a glue table is to help determine which simulated tile the macrotile should resolve into. To facilitate this, we introduce a tile gadget capable of reading an input glue signature from an entry of a glue table and comparing it to another glue signature representing the glues presented to the macrotile. This gadget is described in Figure 16 and is capable of comparing two bit strings of the same length for equality. The result of the comparison is a column of tiles which each encode a whether the corresponding symbols in each string match. These boolean values are then reduces via the AND function to result in a single boolean value which is true if and only if all symbols match. The result can then be used to control the behavior of the tiles on the corresponding output glue signature of the glue table entry. If true, then the output glue signature may be rotated to replace the input glue signature, otherwise the input glue signature will be propagated to the next entry of the glue table for comparison.

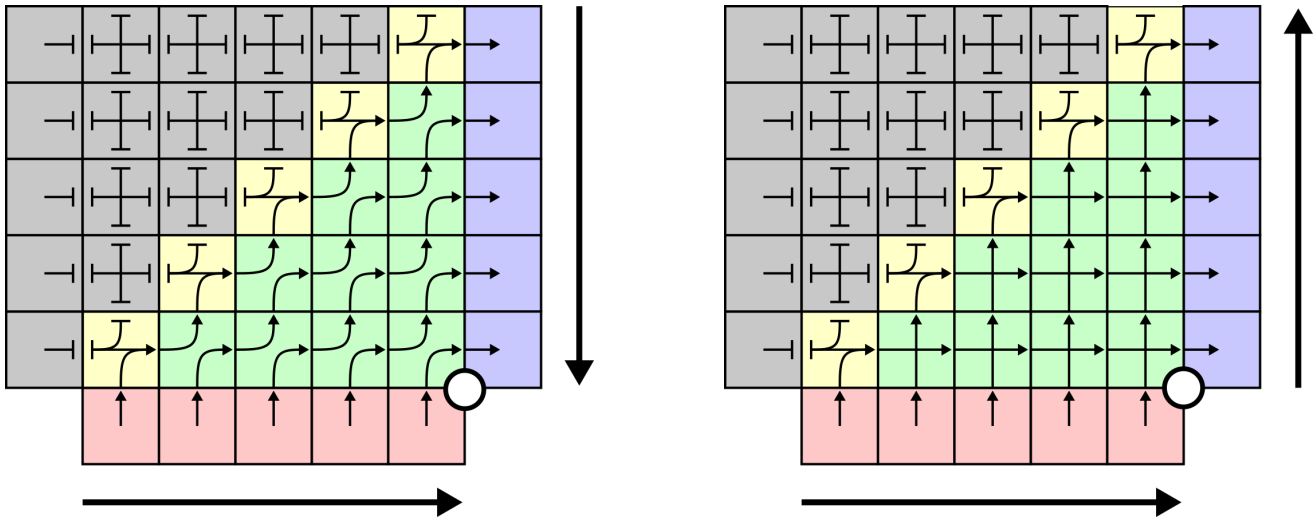
6 A Self-Reproducing aTAM System: Self-Assembly of a Quine

We prove Theorem 3.2 by construction, demonstrating an aTAM quine $\mathcal{Q} = (Q, \sigma, 2)$ for the tileset U that is IU for standard aTAM systems (and its corresponding representation and seed generating functions R and S) given in the proof of Theorem 3.1. Section 3.1 contained a brief overview.

THEOREM 6.1. *There exists an aTAM quine.*

Proof. The remainder of this section consists of the full details of the construction of $\mathcal{Q}$.

6.1 Overview of $\mathcal{Q}$ A high-level, schematic depiction of $\mathcal{Q}$ can be seen in Figure 2a. Here we briefly describe its main components and growth process.



(a) *Cis*-rotation results in the output bits being aligned in the same direction as the input bits. Arrows to the side of the tiles indicate the orientation of the bits with the arrow pointing from most to least significant. Note that both the input and output bits have the MSB nearest the center of rotation.

(b) *Trans*-rotation results in the output bits being oriented in the opposite direction of the input bits. Notice that the input bits have their MSB nearest the center of rotation, while the output bits are the opposite. Both types of rotations can be implemented using the same number of tiles.

Figure 14: Gadgets can be made to rotate bit information encoded on the glues of tiles. After rotation, the resulting bit information will be propagating orthogonally to it's initial direction of propagation. Input bits are represented by red tiles and output bits by blue tiles. Gray tiles indicate tiles with no bit information, but which present strength-1 glues along which the rotation tiles can attach. Lines ending in a T-shape rather than an arrow indicate that a strength-1 attachment is occurring, but there is not bit information being propagated. Arrows indicate the propagation of a bit. The white circle indicates the center of rotation.

Overview of $\mathcal{Q}$:

1. The seed σ consists of a single tile.
2. Q is an IO marked tile set.
3. $Q = T_S \cup T_F$, where T_S is a subset of tile types we refer to as *seed row* tiles, and T_F is a subset of tile types we refer to as *functional* tiles.
4. The tile type of σ is in T_S, and one copy of each tile of the types in T_S attaches to the right of σ to form a hard-coded line of length $|T_S|$ that we refer to as the *seed row*.
5. The glues on the north side of the seed row present a preliminary compressed version of the glue lookup table for Q that contains glue lookup entries for only the tile types in T_F, as well as additional information necessary to complete the tile lookup table and correctly format the sides of the macrotile that forms.
6. The primary functions of the functional tiles that attach to the seed row and grow the macrotile are:
   (a) For each tile in the seed row, compute its compressed glue lookup entry and append that to the glue lookup table encoding so that the table eventually has entries for all tiles in $Q = T_F \cup T_S$.
   (b) Decompress the glue encodings to ensure that all encodings are of the same width.
   (c) Turn the assembly into a square while positioning and formatting the information encoded its perimeter to be consistent with the format utilized by the IU tile set. That is, turn it into a macrotile representing its own seed, with a full glue lookup table of the entire system encoded on the perimeter.

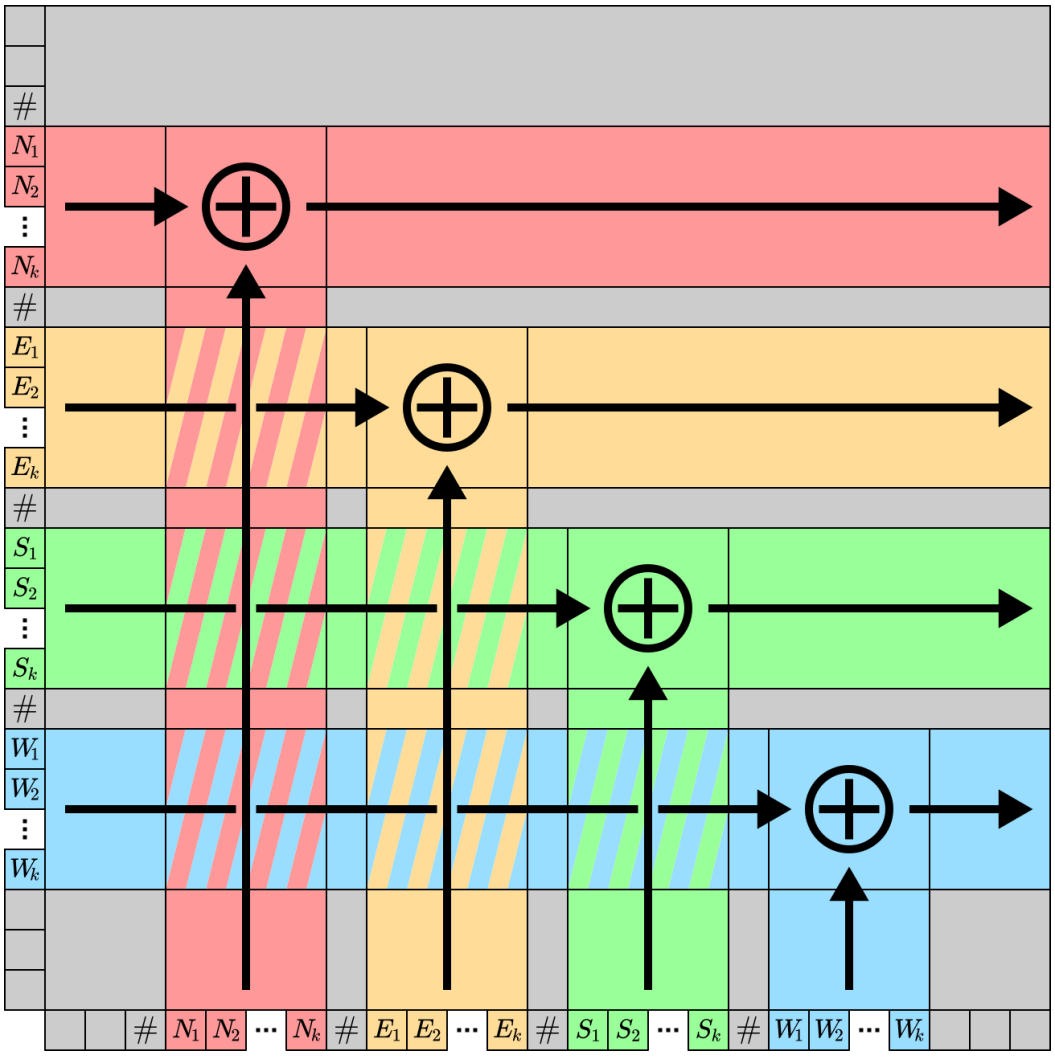


Figure 15: A schematic for the process of combining glue representations in the SW corner of a newly-forming macrotile. The adjacent macrotiles from the west and south present their glue signatures. The macrotile on the west presents an eastward facing glue so its signature only contains non-zero bits after the second glue separator ("#"). Likewise for the south macrotile, it presents a north glue after the first glue separator. These glues are combined into a glue representation encoding both the south and west inputs. This is done using rectilinear growth, initiated from the point of cooperation between the padding regions between the diagonally adjacent tiles. Each of the binary representations in the south signature (or more generically, the counter-clockwise-most) is propagated upwards until it meets with the corresponding binary representations from the east. The binary representations are then XOR'ed together. Binary representations corresponding to different glue directions will be propagated through each other. To determine which direction a binary string corresponds to, the tiles that perform the propagation and combination also keep count of how many separators they have passed. In the end, the result is a new glue signature propagated to the east (more generally, the opposite direction of the clockwise-most cooperating macrotile) with non-zero strings for 2 directions.

In order to understand how the tiles of T_F perform their work, we first describe several sets, values, and encodings that will be used in the construction.

6.2 Glue lookup table encoding Let G_V be the set of all glue labels on any north or south (i.e. vertically-binding) sides of tiles in T_F, and G_H be the set of all glue labels on any east or west (i.e. horizontally-binding) sides

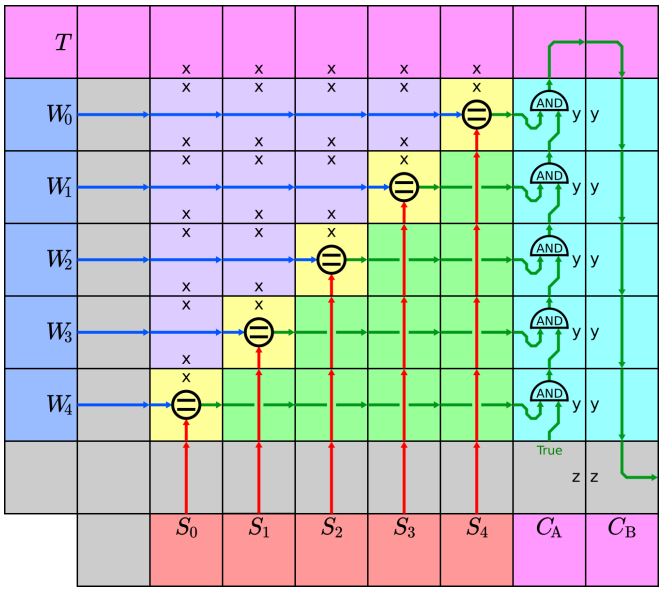


Figure 16: A schematic for a gadget which compares two strings of symbols and determines if they are equal. The string from the west $W_0 \dots W_4$ is compared with the string from the south $S_0 \dots S_4$ by means of tiles which attach in the rectangle spanned between them. These tiles propagate the information from both the west and south strings until they meet in the diagonal. If a symbol of one string matches the corresponding symbol of the other, a boolean signal (either true or false) will be propagated to the east indicating the match. Once the tiles have attached, an upward growing column of tiles will attach along the column of boolean signals and effectively AND all of them together (cyan tiles). This final signal is propagated back down to the south east of the gadget where it can be used to determine how the next gadget behaves. This behavior can be used to read from the glue table by performing a string comparison between each input entry of the glue table with the glue signature representing the current inputs to the macrotile. If the output boolean signal is true, indicating that the glue signature matches the input entry of the glue table, then it is "known" by the macrotile that the corresponding output entry encodes the tile to which the macrotile should resolve.

of tiles in T_F. Let $g_m = \texttt{max}(|G_V|, |G_H|)$ be the size of the largest set, G_V or G_H. Let function $\texttt{BIN} : \mathbb{N} \to \{0,1\}^*$ be defined such that $\texttt{BIN}(n)$ is the (standard) binary representation of natural number n. Thus, $l_g = |\texttt{BIN}(g_m)|$ is the number of bits needed for the binary representation of the size of the largest set, G_V or G_H.

Let $l_s = |T_S|$ be the number of seed row tiles, which is also the length of the seed row. Each adjacent pair of seed row tiles has a strength-2 glue on their abutting west and east sides with a label that is unique to that pair, among all glues in $T_F \cup T_S$. (These will be the only glue labels unique to the seed row tiles, as all other glue labels will also be found on tiles of T_F, allowing them to bind to the seed row tiles.) This requires $l_s - 1$ unique glue labels. If we let G'_H be G_H unioned with these glues, then if we let $g'_m = g_m + l_s - 1$, since $l_s > 0$ we know $g'_m \geq |G'_H|$ and $g'_m \geq |G_V|$, meaning that g'_m is at least as large as the size of the largest set of unique glue labels, the vertical or horizontal glue labels. We define $l_{pg} = |\texttt{BIN}(g'_m)|$, the length of the binary representation of g'_m, and call it the *padded glue length*.

Given a set of glue labels G, let $\texttt{LEX}(G)$ be a lexicographic ordering of the glue labels in G. To create the glue encodings for our construction, we use the alphabet $\Sigma_{GE} = \{0,1\}$ and define the function $\texttt{PAD\_BIN} : \mathbb{N} \times \mathbb{N} \to \{0,1\}^*$ such that $\texttt{PAD\_BIN}(n, l)$ is the string $s \in \{0,1\}^*$ consisting of the binary representation of n padded on the left with the number of 0s needed to make $|s| = l$, with the requirement that $|\texttt{BIN}(n)| \leq l$. For each $0 \leq i < |\texttt{LEX}(G_V)|$, let g_i be $\texttt{PAD\_BIN}(i, p)$. To create the glue encoding for a vertical glue (i.e. one on a north or south glue face) we use the index i of its label in $\texttt{LEX}(G_V)$ to get $\texttt{PAD\_BIN}(i, l_{pg})$. Then, if the glue has strength 1, we prepend a 0 to $\texttt{PAD\_BIN}(i, l_{pg})$. If it has strength 2, we instead prepend a 1. We define the glue encodings for G'_H similarly, noting that the glue encodings of any two glues in the same set, G_V or G'_H, will be unique, but two glues in different sets may have the same encoding. Furthermore, since l_{pg} is based on g'_m we know that l_{pg} is guaranteed to be long enough to contain the bits of any glue encoding in G_V or G'_H and thus any glue in $Q = T_F \cup T_S$, with possible leading 0s for padding. Therefore, all glue encodings will be of the same length, $l_{pg} + 1$. Given a tile type t, direction $d \in \{N, E, S, W\}$, and $io \in \{\texttt{IN}, \texttt{OUT}\}$, we define $\texttt{GLUE\_ENC}(t, d, io)$ as a function such that, if $io = \texttt{IN}$ and the glue on side d of t is an input glue, or $io = \texttt{OUT}$ and the glue on side d of t is an output glue, returns the glue encoding of that glue in the specified format. Otherwise, if the value of io does not match the glue's input/output status, a string of $l_{pg} + 1$ zeros is returned. One additional value that we will define for use later is $l_p = l_{pg} - l_g$. That is, l_p is the length of padded glues minus the length of the longest encoding needed for a glue in G_V or G_H (whichever requires the longest). Note the use of G_H and not G'_H, so this does not include the glues specific the T_S, which contain the (overwhelmingly) largest proportion of all unique glue labels in Q.

For the glue lookup entries of our construction, we use the alphabet $\Sigma_{GLE} = \{0, 1, \#, |\ 1, |\ 2\}$. For each $t \in Q$, the glue lookup entry for t consists of the concatenation of the following three strings (which consist of the concatenation of the encodings of input glues, two separator symbols, then concatenation of the encodings of the output glues):

'$\#\texttt{GLUE\_ENC}(t, N, \texttt{IN})\#\texttt{GLUE\_ENC}(t, E, \texttt{IN})\#\texttt{GLUE\_ENC}(t, S, \texttt{IN})\#\texttt{GLUE\_ENC}(t, W, \texttt{IN})$'

'$|\ 1\ |\ 2$'

'$\#\texttt{GLUE\_ENC}(t, N, \texttt{OUT})\#\texttt{GLUE\_ENC}(t, E, \texttt{OUT})\#\texttt{GLUE\_ENC}(t, S, \texttt{OUT})\#\texttt{GLUE\_ENC}(t, W, \texttt{OUT})$'

Given a tile type $t \in Q$, we define $\texttt{GLE}(t)$ as the function that returns the glue lookup entry for t using that format.

For the glue lookup table of our construction, we use the alphabet $\Sigma_{GLT} = \{0, 1, \#, |\ 1, |\ 2, \$\}$, and the table is the string '$\$\texttt{GLE}(t_0)\$\texttt{GLE}(t_0) \ldots \$\texttt{GLE}(t_{i-1})$ for $0 \leq i < |Q|$ and t_i the ith tile type of Q (i.e. the glue lookup entry of each tile type, with a \$ symbol prepended to each).

6.3 Macrotile side encoding Since only the north and east sides of the seed tile type, t_σ, have non-null glues, only the north and east sides of the macrotile representing σ will encode glues and the glue lookup table. The glues on those sides of the macrotile represent 7 distinct regions, each contained between a pair of lexicographically adjacent pairs of *delimiter symbols* taken from the set $\{\texttt{B0}, \texttt{B1}, \texttt{B2}, \texttt{B3}, \texttt{B4}, \texttt{B5}, \texttt{B6}, \texttt{B7}\}$. We will refer to the layout of the symbols and regions on the north side of the macrotile as follows (and for the east side, g_N is replaced with g_E):

$\texttt{B0}\langle\texttt{blank}_{out}\rangle\texttt{B1}\langle g_N\rangle\texttt{B2}\langle\texttt{blank}_{in}\rangle\texttt{B3}\langle T\rangle\texttt{B4}\langle\texttt{blank}_{in}\rangle\texttt{B5}\langle g_N\rangle\texttt{B6}\langle\texttt{blank}_{out}\rangle\texttt{B7}$

The strings between the pairs of delimiters are defined as follows:

$\langle\texttt{blank}_{out}\rangle$ = a string of blank symbols (i.e. '_') of length l_{out}, where l_{out} will be defined later.

$\langle g_N\rangle$ = '$\#\texttt{GLUE\_ENC}(t_\sigma, N, \texttt{OUT})\#0^{l_{pg}+1}\#0^{l_{pg}+1}\#0^{l_{pg}+1}$'. That is, a string consisting of the glue encoding of

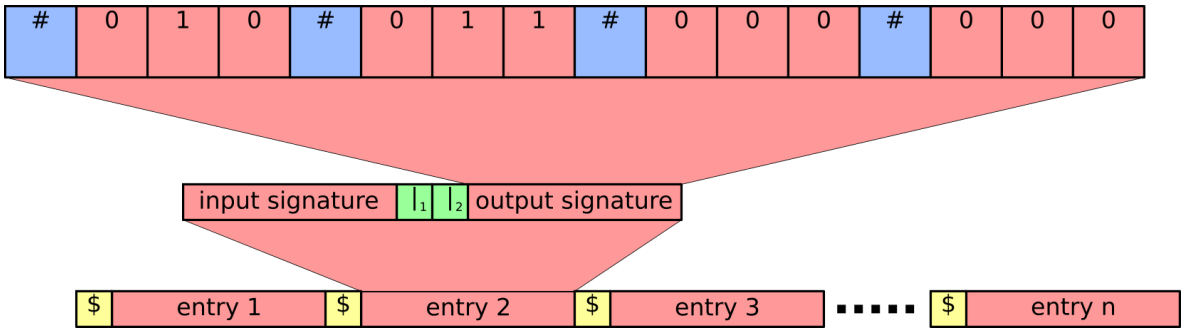


Figure 17: Example depicting the format of the glue lookup table. The glue lookup table (bottom) is composed of a list of glue lookup entries separated by $ characters. Each glue lookup entry (middle) is composed of the input and output signatures of the corresponding tile separated by the two characters | | 1 and | 2. The input and output signatures have the same format, and an example output signature (top) is shown with output glues on the N and E sides (since the W and S glues are all 0s, or *null*). Since each of the N and E glue representations begin with a 0, they each represent a strength-1 glue, and their labels are encoded as 10 and 11, respectively.

the north glue of t_σ followed by "empty" glue encodings for the other three directions (i.e. each is a string of $l_{pg}+1$ zeroes).

$\langle \texttt{blank}_{in} \rangle$ = a string of blank symbols (i.e. '_') of length l_{in}, where l_{in} will be defined later.

$\langle T \rangle$ = the glue lookup table for Q.

On the east side of the macrotile will be the same information, presented from top to bottom, with the slight difference that in the regions between `B1` and `B2`, and between `B5` and `B6`, the east glue of t_σ will be encoded instead of the north, i.e. '$\#0^{l_{pg}+1}\#\texttt{GLUE\_ENC}(t_\sigma, E, \texttt{OUT})\#0^{l_{pg}+1}\#0^{l_{pg}+1}$'.

6.4 Seed row encoding The labels on the glues on north sides of the tiles of the seed row encode a truncated, compressed, and slightly modified version of the information that will eventually be presented on the north and east sides of the macrotile. We will refer to the layout of the symbols and regions on the north side of the seed row as follows:

`B0!`$\langle$`ctrpad`$\rangle$`B1`$\langle g_N, g_E \rangle$`B2`$\langle$`gluepad`$\rangle$`B3`$\langle T \rangle$`B4`

The strings between the pairs of delimiters are defined as follows:

$\langle$`ctrpad`$\rangle$ = the binary string representing $g_m + 1$ padded to length l_{pg} with 0s on the left, then reversed in direction so that the least significant bit is on the left.

$\langle g_N, g_E \rangle$ = '$\#\texttt{GLUE\_ENC}(t_\sigma, N, \texttt{OUT})\#\texttt{GLUE\_ENC}(t_\sigma, N, \texttt{OUT})\#0^{l_{pg}+1}\#0^{l_{pg}+1}$'. That is, a string consisting of the glue encoding of the north glue of t_σ followed by glue encoding of the east glue of t_σ, then 2 "empty" glue encodings for the other two directions (i.e. each is a string of $l_{pg}+1$ zeroes).

$\langle$`gluepad`$\rangle$ = a string of 0s of length l_p.

$\langle T \rangle$ = the compressed glue lookup table for the tiles of T_F.

The "`B0!`" symbol is a special symbol used to initiate growth and later replaced by the "`B0`" delimiter symbol.

The northern glues of the tiles of T_S in the $\langle T \rangle$ region present encodings of all of the tiles of T_F, and since there are g_m symbols in the largest set, G_V or G_H, which contain all of the vertical and horizontal glue labels of T_F, respectively, then the largest value of a glue encoding used to encode the tiles of T_F is g_m (since the value 0 is reserved for the null glue). As previously mentioned, none of the vertical glue labels of the seed tiles are unique to tiles of T_S (i.e. one or more tiles of T_F also use each). Additionally, the east glue label of the easternmost seed row tile is `FILL_BOTT`, which is shared by another tile in T_F (and therefore already in G_H), and the south sides of the seed row tiles have no glues. Therefore the only new glue labels that need to be encoded in order to create the glue lookup entries for the tiles of T_S are those between pairs of seed row tiles. As stated, there are $l_s - 1$ of

these (where $l_s = |T_S|$). In order to allow the tiles of T_F to create the glue lookup entries of the tiles of T_S, the value $g_m + 1$ is encoded into the $\langle$`ctrpad`$\rangle$ region between `B0` and `B1`. This will be the value of the glue encoding of the first new glue representation.

In order to make the seed row more compact, requiring many fewer tiles for T_S, we compress the glue encodings contained in the glue lookup table which encodes T_F. This is done in two ways: (1) any glue encoding which needs to represent an empty glue location is represented by a single 'n' character (rather than the string of 0s of length l_{pg} that will later be necessary), and (2) for the glue encoding of each non-empty glue, starting from the second leftmost bit (since the leftmost bit represents the strength of the glue), l_p bits are replaced by a single 'p' character. Those bits are guaranteed to all be 0s by the definition of l_p, since it $l_p = l_{pg} - l_g$, meaning that it is the length of a fully padded binary representation of a glue minus the length of the longest encoding in G_V and G_H, which includes any glue being encoded for T_F. So, the additional bits (which are only needed for the encoding of horizontal glues of seed row tiles) can be replaced by the 'p' character and since $\langle$`gluepad`$\rangle$ contains a string of l_p 0s, each such 'p' can later be replaced by l_p 0s. (Note that the only glue encodings that are not compressed in this way are those in the $\langle g_N, g_E \rangle$ region due to the physical layout of the information later used to decompress glue encodings.) The "decompression" is necessary because the IU tileset U requires that all fields representing glues on the sides of a macrotile to be of the same width.

6.5 Building the macrotile from the seed row In order to grow and transform the information encoded along the north of the seed row into the information needed for the sides of the macrotile, the following phases of assembly occur:

1. Phase 1: For each tile in the seed row, the northernmost row is extended by a fixed number of tiles that consist of a preliminary representation of that seed tile, called a *compressed glue lookup entry.*
2. Phase 2: Every 'p' symbol is replaced by a string of l_p 0s.
3. Phase 3: Every 'n' symbol is replaced by a string of $l_{pg} + 1$ 0s.
4. Phase 4: The `ctr` symbols in the seed row tile templates are replaced by appropriate values of the counter encoded in $\langle$`ctrpad`$\rangle$.
5. Phase 5: The symbols in the regions between `B0` and `B1`, and between `B2` and `B3`, are each replaced by a '_' symbol.
6. Phase 6: The contents of the regions between (a) `B2` and `B3`, (b) `B1` and `B2`, and (c) `B0` and `B1` are copied to extend the right side of the row into regions between (a) `B4` and `B5`, (b) `B5` and `B6`, and (c) `B6` and `B7`, respectively.
7. Phase 7: The rectangular assembly created by the previous phases is turned into a square macrotile with correct representations on the N and E sides of the glues of t_σ and all information rotated and spaced appropriately for simulation of the system by the IU tile set.

We now give a brief overview of each phase. Note that for Phases 1-6, all growth is done in a zig-zag manner after the seed row. The seed row grows from left to right, and so does row 1 directly across its north. Then, row 2 and all subsequent even numbered rows grow from right to left. Row 3 and all odd rows grow from from left to right. The right-to-left growing rows always start immediately north of the furthest rightmost tile(s) and stop in the column immediately north of the seed tile (at x-coordinate 0). The left-to-right growing rows always start at x-coordinate 0 and either stop above the furthest rightmost tile(s) or extend the row by some constant number of tiles (depending on the phase) beyond the previously rightmost tile(s).

6.5.1 Phase 1: adding compressed glue lookup entries for the tiles of T_S In this phase, the compressed glue lookup entry for each tile of the seed row is appended to the right end of the seed row. Figure 18 shows an example of the compressed lookup entry added for the seed tile.

1. For $0 < i < |T_S| - 1$, let $t_{s_i} \in T_S$ be the ith tile of T_S and also the ith tile from the left of the seed row once it has completed growth.

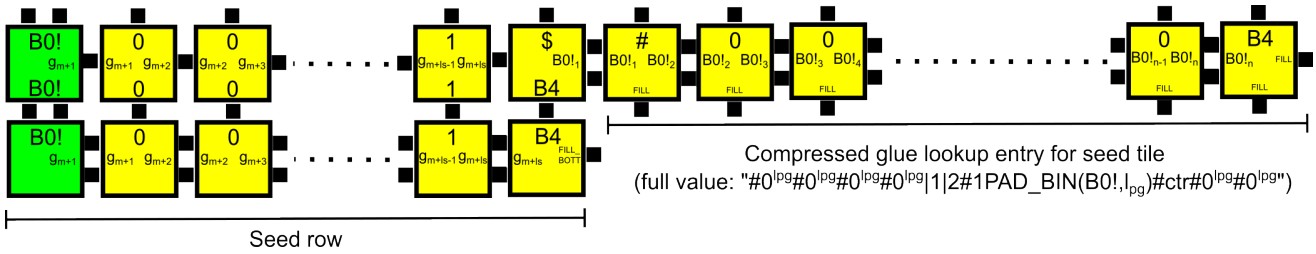


Figure 18: Example depiction of the seed row (bottom) and the compressed glue lookup table entry added for the leftmost (i.e. seed) tile (top). (Not all glue markings are shown.)

2. During the growth of this phase, which is performed by the tiles of T_F, the full definition of each t_{s_i} is inferred from its location in the seed row. For each, its northern glue can be read directly by a tile from T_F attaching to its north since the seed row tiles are carefully designed so that their northern glues only consist of the following symbols:

   $\{\texttt{B0!}, \texttt{B1}, \texttt{B2}, \texttt{B3}, \texttt{B4}, \$, |_1, |_2, \#, n, p, 0, 1\}$.

   Therefore, the tiles of T_F can be created before the tiles of T_S but still be guaranteed to be able to correctly bind to them and infer their definitions. The tile type of t_{s_0} only has that northern glue and a strength-2 glue on its east side that allows a tile of type t_{s_1} to bind, and its south and east sides contain the null glue. All other tiles in the seed row except for the rightmost, i.e. t_{s_i}, are similar except that they each also have a western glue that allow them to bind to $t_{s_{i-1}}$. The rightmost tile of the seed row, of type $t_{s_{|T_S|-1}}$, only has such a western glue, but its eastern glue is of strength-1 and has a label of '`FILL_BOTT`' (which will be explained later).

3. The strength-2 glues between each pair t_{s_i} and $t_{s_{i+1}}$ are assigned consecutive glue encoding values beginning from g_{m+1}, which is the value encoded in $\langle\texttt{ctrpad}\rangle$.

4. For example, the tile type of t_{s_0} is known to be a special case with a strength-2 glue on its north whose label is `B0!` and it has one other non-null glue, on its east side, that is of strength-2 and whose label is assigned the glue encoding g_{m+1}. Since it is the seed tile and thus has no input, its north glue is strength-2 with the label `B0!` (which gets mapped to index number 1), and its eastern glue is assigned the current counter value (denoted by the `ctr` symbol) its compressed glue lookup entry is the string '$\#0^{l_{pg}}\#0^{l_{pg}}\#0^{l_{pg}}\#0^{l_{pg}}|_1|_2\#1\texttt{PAD\_BIN}(\texttt{B0!}, l_{pg})\#\texttt{ctr}\#0^{l_{pg}}\#0^{l_{pg}}$'. For another example, we'll describe the entry for a tile t_{s_i}, for $0 < i < |T_S| - 1$, assuming it has a north glue of strength-1 encoding a symbol $x \in \{\texttt{B1}, \texttt{B2}, \texttt{B3}, \texttt{B4}, \$, |_1, |_2, \#, n, p, 0, 1\}$ (which is the full set of labels of the northern glues of the tiles of T_S other than the leftmost and rightmost, which are `B0!` and `B4`, respectively). Then, the glue lookup entry for t_{s_i} is '$\#0^{l_{pg}}\#0^{l_{pg}}\#0^{l_{pg}}\#\texttt{ctr}|_1|_2\#0\texttt{PAD\_BIN}(x, l_{pg})\#\texttt{ctr}\#0^{l_{pg}}\#0^{l_{pg}}$'.

5. Since the north glue of any tile in T_S is one of those in the given set and the rest of the tile's definition is fixed with the caveat that the value used to encode the horizontal glues increases by one for each glue moving to the right and a single generic variable symbol `ctr` can be used to represent all of these values (to be explained in a later phase), there are a fixed number of possible compressed glue lookup entries of fixed width. Thus, the hard-coded set of tiles needed for each possible northern symbol of a seed row tile are created as part of T_F, before the tiles of T_S are generated (therefore allowing the tiles of T_S to encode them in their northern glues).

6. Each row of Phase 1 that grows from left to right reads and stores the northern glue label x of the tile that currently has the marking symbol `ext` (which starts on the leftmost tile), moves the `ext` symbol one tile to the right, then grows to the far right end of the row below it, whose rightmost tile has the glue label `B4`. At that location, the tiles that encode the compressed glue lookup entry for symbol x to attach and extend the row further to the right. The location with the `B4` symbol is changed to have a $ symbol, and the `B4` symbol is placed at the rightmost end of the newly extended row.

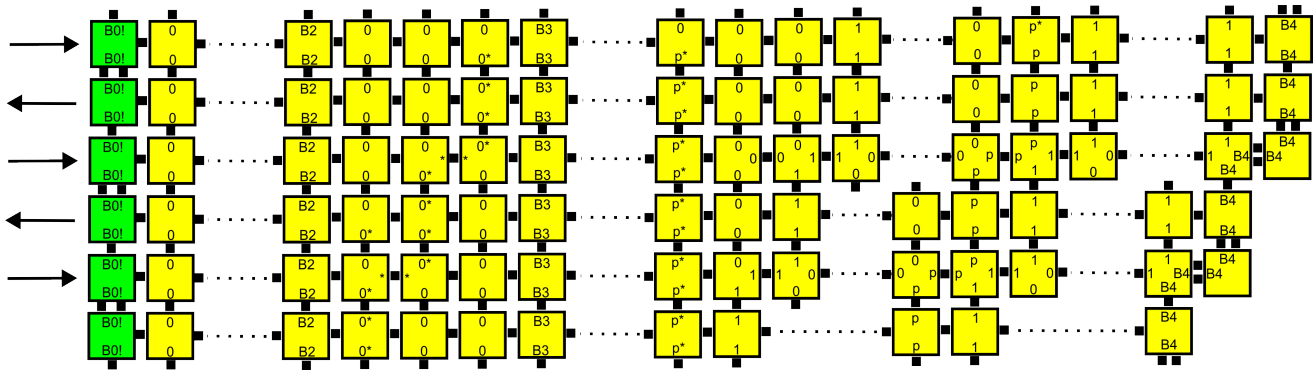


Figure 19: High-level depiction of an iteration of the main loop of Phase 2, i.e. inserting a string of 0s (whose length is equivalent to the number of 0s between the B2 and B3 delimiters) into the location of the leftmost p symbol. (Note that, to highlight the main logic, not all glues are fully shown.) At the beginning of an iteration (the bottom row), the leftmost 0 following B2 is marked (here with a "*" symbol), and the leftmost p is also marked. The first row of the iteration grows above it, from left to right, and moves the marker on the 0 one position to the right, then inserts a tile representing a 0 immediately following the marked p. This requires the inserted location and the rest of the tiles of the row to pass the original value of each column to the right and to change the value of the column to the value received from their left, growing the row by one tile. Right-to-left growing rows simply copy values upward. Subsequent left-to-right growing rows continue the process until the final 0 before B3 is encountered, at which point the marker for the 0s is removed and that row replaces the marked p with a 0, then marks the next p encountered to the right. In this example, three 0s are between B2 and B3, so the marked p is replaced by three 0s. To begin the next iteration, the first 0 after B2 is marked. Iterations continue until no p symbols remain in the row.

7. Having completed growth of that row, adding the encoding of one more tile of T_S, the next row grows all the way back to the left and initiates growth of the next left-to-right now that continues the process. Once the marker ext passes the column in which the seed row ended (whose location is preserved via another special marker symbol), Phase 1 is complete since every tile of T_S will now have a compressed glue lookup entry for it added to the tile lookup table.

At the end of this phase, there is a (compressed) glue lookup entry for every tile in Q.

6.5.2 Phase 2: replacing p symbols with strings of l_p 0s Phase 2 executes a relatively simple procedure. It begins by marking the location of the leftmost p. Then, for each 0 in the region between B2 and B3, except for the final 0, it "inserts" a 0 after the marked p. It does this by growing a row to one position right of the marked p, setting the north glue of the tile at that location to 0 while encoding the symbol currently at that location in its east glue. The tile to the east places uses that symbol for its northern glue and propagates the current value for that location to the right. This occurs until the B4 symbol is encountered at the end of the row, at which point it is most right to a tile that extends the row rightward by one tile. For the final 0, instead of inserting a 0, it simply replaces the marked p by 0. In this way, the location of the marked p has a string of l_p 0s inserted in its place, and by moving the mark to the next p to the right after completing each such loop, that happens for all p symbols. This results in all glues with the "padding" marker being decompressed to the full, fixed length required for all glue encodings. Figure 19 shows a high-level example of one such loop iteration.

6.5.3 Phase 3: replacing n symbols with strings of $l_{pg} + 1$ 0s Phase 3 is performed in almost the exact same manner as Phase 2, but with the number of positions between B0 and B1 being used to determine how many 0s to insert for each n symbol. Irrespective of the actual bit values between B0 and B1, for each a 0 is inserted, and for the final bit two 0s are inserted so that the n symbol will be replaced by a string of 0 symbols of length $l_{pg} + 1$, since each glue label is encoded with l_{pg} bits and the extra 0 bit is in the location denoting the strength of the glue. Recall that this glue encoding, composed of all 0s, represents the special case of the null glue.

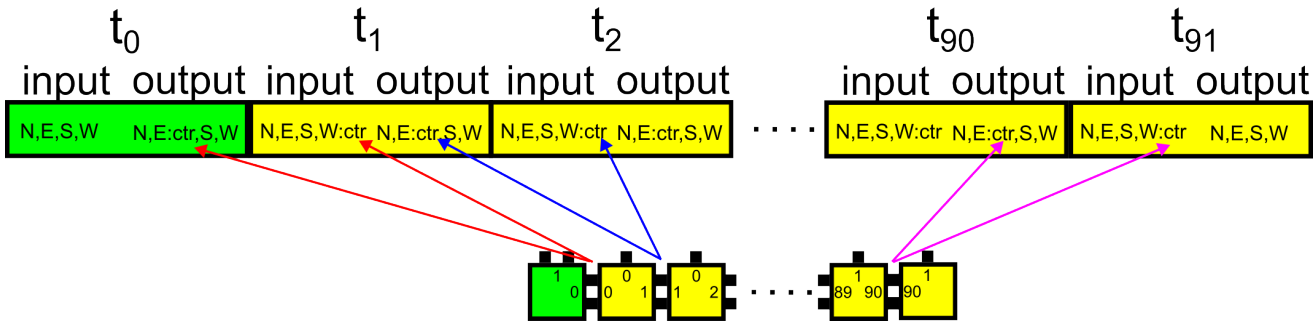


Figure 20: Schematic example of seed row tiles (bottom) and their corresponding compressed glue lookup entries (top). The only instance of the `ctr` symbol in the entry for t_0 will be as its output glue to the east. The entry for t_1 has two instances of `ctr` with the left representing its input glue to the west and the second its output glue to the east. Note that the leftmost two instances of `ctr` in the table refer to the same glue, shared by t_0 and t_1. Therefore, the same value, which is the current value of the counter whose bits are represented in the region between `B0` and `B1`, is put into both locations. Then the next two instances of `ctr` also refer to the same glue, so once the counter is incremented to create a new unique encoding value for that glue, the counter's value can then be put into both locations. This pattern continues until the final, rightmost glue between seed row tiles.

6.5.4 Phase 4: replacing the `ctr` symbols with counter values Phase 4 is also performed in a manner very similar to the previous two phases, with just a few notable differences. For $0 \leq i < |T_S|$, let g_{s_i} be the label of the horizontal glue shared by two adjacent seed row tiles such that it is the ith such glue from the left (e.g. g_{s_0} is the label of the glue shared by leftmost seed row tile and its neighbor immediately to the right). The compressed glue lookup entries for the seed row tiles (added in Phase 1) are located from left to right in the same order as the seed row tiles themselves, and within those the glues used as input are listed to the left of those used as output. The leftmost seed row tile has no input since it is t_σ, the seed of the system, and every tile to the right of that tile has a single input glue on its west side. Except for the rightmost seed row tile, each has an output to its east. Since compressed glue lookup entries encode all such glues simply using the symbol `ctr`, and each pair of adjacent edges between seed row tiles share the same glue, the pattern exists where the leftmost two instances of `ctr` in the table refer to the same glue as each other, the next two instances to the same glue as each other, etc. This pattern can be seen in Figure 20. Since the glues are encoded using values that are each one larger than the previous, from left to right, in this phase all that is required is that the current value of the counter bits is inserted into the locations of both of the two leftmost `ctr` symbols, and then the counter is incremented by one and the process repeated until all copies of `ctr` have been replaced.

Recall that the symbols representing the bits of the counter in the seed row are in reverse order, with the least significant bit being on the left. Since the bits are copied from left to right, and each inserted immediately following a `ctr` symbol, the copied values are again reversed, to be in the correct order. Finally, once all counter bits have been copied to the right of a `ctr` symbol, the `ctr` symbol is replaced by a 1 symbol. This is in the position of the bit that specifies the glue's strength, and is a 1 because all such glues are of strength-2.

6.5.5 Phase 5: Blanking out spacer regions Phase 5 is the simplest phase. Its purpose is simply to turn all symbols in the current "spacer" regions ($\texttt{blank}_{out}$ between `B0` and `B1`, and $\texttt{blank}_{in}$ between `B2` and `B3`), into blank, i.e. "_", symbols. This is done by a single row that grows left to right that detects when it is growing over one of those regions and presents only "_" symbols to the north until leaving the region. Once completed, a row grows back right-to-left to reset and allow for the next phase to begin.

6.5.6 Phase 6: Copying regions to the right Phase 6 also performs a relatively simple task. Namely, it copies the contents of the $\texttt{blank}_{out}$, g_N, g_E, and $\texttt{blank}_{in}$ regions so that they each have a copy on the right side of the glue lookup table. The ordering and a schematic depiction can be seen in Figure 21.

6.5.7 Phase 7: Squaring the Quine (chmod +x quine.atam) In the previous stages, the rightmost tiles of each row are given a strength-1 glue labeled `FILL`, except for those of the first row, which are given a strength-1

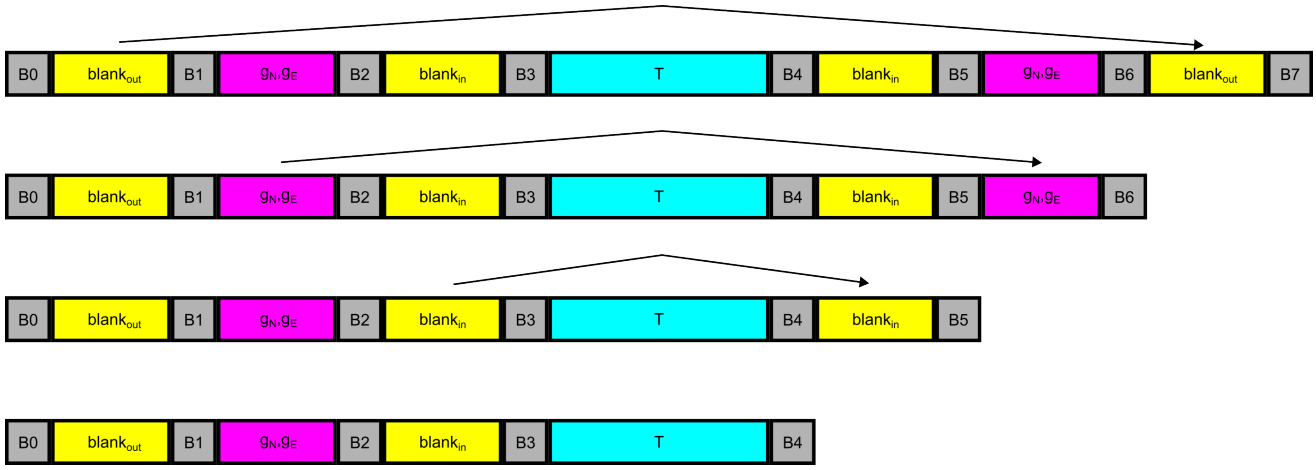


Figure 21: Schematic depiction of Phase 6 of the quine creation, the copying of 3 regions to the right. From bottom to top, the three leftmost regions are copied to the right side. Although the ordering in which the regions are copied is from right to left, the contents of each region are copied from left to right so that they end up in the same ordering on both sides.

glue labeled `FILL_BOTT`. Additionally, every tile that attaches via a strength-2 glue on its west, and thus has no input glue on its south, has a strength-1 glues labeled `FILL` on its south. This allows two tile types, the "filler" type with strength-1 glues labeled `FILL` on all four sides, and the "filler bottom" type with a strength-1 glue labeled `FILL` on its north and strength-1 glues labeled `FILL_BOTT` on its east and west sides, to cause the otherwise diagonally slanted assembly to fill out into a rectangle.

An example assembly resulting after the first 6 phases, produced in the WebTAS simulator [18] (with a seed row of reduced size due to the large scale factor), can be seen in Figure 22. In order for this assembly to become a quine and act as a macrotile seed for tileset U (which is the IU tileset for standard aTAM systems), it needs to be in the shape of a square and have its perimeter glues in the format of a macrotile utilized by U. Note that, as this quine construction is a component our first fractal construction (i.e. Theorem 3.5, the strict self-assembly of a DSSF), the way in which we create the square macrotile is slightly more complex than would be necessary solely for the proof of Theorem 3.2. However, this is done to simplify the final construction for Theorem 3.5. Therefore, in this section we will describe the basic features needed for the current proof, and the additional complexities will be further explained in the proof of Theorem 3.5.

Figure 23 shows an overview of the process that completes the macrotile formation. The rectangular assembly formed by the first 6 phases has the information required for a macrotile used by U on its north, including both the glue information that will ultimately need to be on the north, g_N, and that for the east, g_E (since those are the two sides of the seed tile type t_σ that are non-null). This information is first rotated using a standard string rotation gadget so that it's presented to both the north and the east. After that, 4 square frames are grown using loops of binary counter gadgets that act as the 4 corners of the macrotile (red and yellow). The size of each frame is dictated by an initial counter value and may be chosen arbitrarily (to be discussed more for the proof of Theorem 3.5) as long as it is greater than the height of the rectangular assembly. Referring to that size as X, $O(\log(X))$ tile types are created for the counters. The space inside of each of the frames is empty except for solid squares of tiles that grow from each interior corner (green). The side length of these green squares, which must be $\leq X/2$, is controlled by an additional binary counter made from hard-coded tiles (also discussed more later). Referring to that size as Y, $O(\log(Y))$ tile types are created for these regions. The information for the north and east sides, rotated from the rectangle, is then propagated along the square frames so that it is centered along the north and east faces of the resulting square macrotile. Additionally, generic filler tiles grow outward to the west side, and from the south of the grey rectangle to the south side.

Upon reaching the ends of the counters, the growths to each side initiate growth of a final row of tiles that forms the outer later of the macrotile. On the south and west sides, the tiles of these rows expose no glues to the south and west, respectively, making the south and west sides of the macrotile blank, representing no output

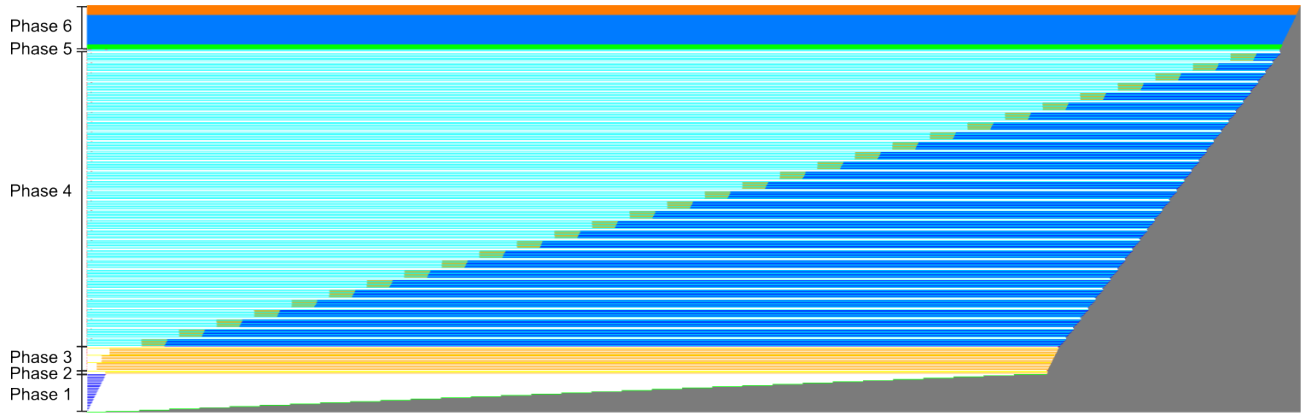


Figure 22: The rectangular assembly of the quine construction after the first 6 phases of growth (with a reduced set of tiles from T_F encoded by the seed row tiles due to the size of the full assembly, so phases aren't exactly to relative scale). Growth is from the bottom upward, and phases are marked. Phase 1: the compressed glue lookup entry of each tile of the seed row (the relatively very short portion to the left of the grey portion at the bottom) is appended. Phase 2: Each 'p' symbol is replaced. (Note that the example seed row used has few p symbols, but there would actually be many more.) Phase 3: Each 'n' symbol is replaced. Phase 4: The '`ctr`' symbols are replaced. Phase 5: The "spacer" regions are replaced with blanks. (Phase 5 is a single pair of rows, which is not very visible at this scale.) Phase 6: The regions to the left of the glue lookup table are copied to the right. The grey portion is composed of tiles of the "filler" type, except for the bottom row which is composed of the "filler bottom" type.

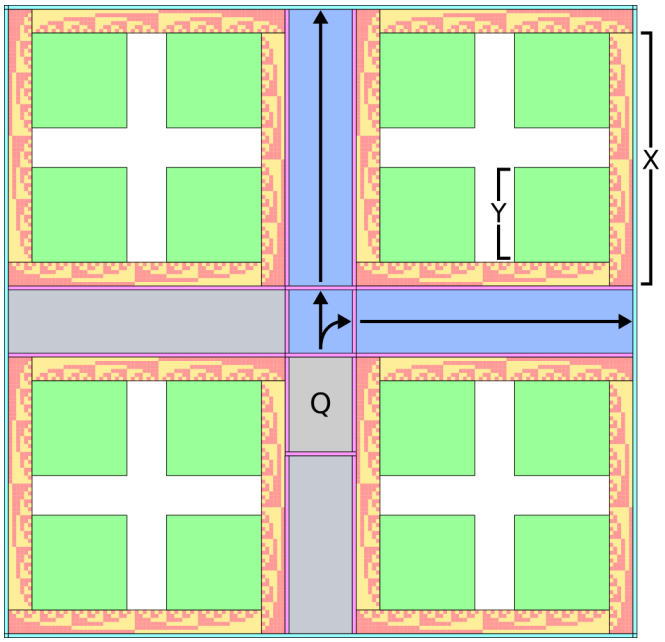


Figure 23: Schematic depiction of the formation of a macrotile square from the rectangular assembly formed after the first 6 phases (shown in grey). The dimension X, counted by binary counters, can be set to an arbitrary value as long as it is greater than the height of the grey rectangle, and the dimension Y can be set to be any value between 0 and $X/2$. The settings of those values will be important for the construction of Theorem 3.5.

glues for those sides. On the north and east sides, the following occurs during the growth of that final row:

1. During the growth of the row along the north, the entries for the east glue, g_E, in the regions between the B1 and B2 delimiters and B5 and B6 delimiters, are changed to be all zeroes, while the entries for g_N are left intact. During the growth of the row along the east, the corresponding entries for g_N are changed to be all zeroes, while the entries for g_E are left intact.
2. The outward facing glues at the locations of the B0 and B7 delimiter symbols are changed to blank symbols (i.e. '_') of strength 1, and as growth proceeds outward from the middle regions (blue in Figure 23) along the counters (orange and yellow in Figure 23), all outward facing glues are blank symbols of strength 1.
3. Other than the tiles placed in these outer rows, the tiles of tileset Q use IO markings that make them incompatible with the tiles of U. Specifically, Q uses the set $\texttt{IO}_{s2} = \{\vee\vee, <<, \wedge\wedge, >>\}$, while U uses the set $\texttt{IO}_{s1} = \{\vee, <, \wedge, >\}$. (This is done to make the constructions modular and ensure that there is no possibility for unintended tile attachments.) In order to make the terminal assembly of $\mathcal{Q}$ compatible with the tiles of U, the IO markings of the outward facing glues of the tiles of these rows use $\texttt{IO}_{s1}$.
4. Once a row has completed growth along an (orange and yellow) counter, it waits until the row of the side with which it shares the corner to complete. At that point, the final two tiles of those sides cooperate to place the corner tiles.
   (a) On the corner tile of the northwest corner, the northern glue is a strength-2 glue with label B0, and its western glue is null.
   (b) On the corner tile of the northeast corner, its northern glue is a strength-1 glue with label B7, and its eastern glue is a strength-2 glue with label B0.
   (c) On the corner tile of the southeast corner, its eastern glue is a strength-1 glue with label B7, and its southern glue is null.
   (d) On the corner tile of the southwest corner, both its southern and western glues are null.

Upon completion of the outer row, the assembly is terminal. The outer row essentially causes the $\texttt{blank}_{out}$ regions to be expanded to encompass the dimensions of the counters, presents the full and correct information needed by a macrotile of system using the tiles of U to simulate $\mathcal{Q}$ (especially the glue encodings for the corresponding sides and the glue lookup table), has the IO markings that are compatible with the tiles of U, and has strength-2 glues exposed along the north and east sides that represent strength-2 glues of t_σ. As such, the terminal assembly correctly seeds itself with respect to U (and representation function R and seed generation function S).

The final point that must be shown is that $\mathcal{Q}$ is a standard aTAM system (which is the class of systems that U is IU for). Recalling that all of the tiles of Q are IO-marked, both facts follow immediately for the growth of phases 1 through 6, which consist solely of (1) zig-zag growth, and (2) additions of the "filler" and "filler bottom" tiles. For the zig-zag growth, all tile attachments are via a single strength-2 input glue or two diagonally adjacent strength-1 input glues, and every tile side with a non-null glue is used as either an input or output glue, meaning that no mismatches occur. Furthermore, careful design of the subsets of tiles to be specific for each phase easily ensures that no two tiles have the same set of input glues. From these facts, it follows that the zig-zag growth is also directed. There are only two filler tile types, "filler" and "filler bottom", and both have two strength-1 input glues, one on each of their north and west sides. The "filler" type has strength-1 output glues on its east and south, and the "filler bottom" only on its east. This means that all attachments are via diagonally adjacent input glues, there can be no mismatches, and all growth is directed. Finally, the growth of phase 7, in which the rectangular assembly grows into a square macrotile also follows our requirements for a standard TAS. The tiles attach in a directed fashion. The square frames of side length $X + \log_2(X)$ are made from standard binary counter gadgets that were easily made to be directed and without mismatches. The interior corner squares also use binary counter gadgets to control their size and are seeded from a single glue presented on the interior corners by the row of tiles that seeds the frame counters. This is the only glue that is shared between the interior squares and the frame (all others being the null glue) to avoid mismatches. The tiles that fill up the space between the frames and propagate the simulation information from the quine also have the null glue on the clockwise-most side to avoid mismatches when interfacing with the opposite side frame.

Thus $\mathcal{Q}$ is a quine and Theorem 3.2 is proven. □

6.6 Breaking circular dependencies in tile type creation Here we discuss a few technical details about the creation of the tile sets T_F and T_S, which have slight circular dependencies, and how those are handled.

The high-level algorithm for generating T_F and T_S is the following:

1. Initialize the value for the full, final width of glue representations $l_{pg} = 1$ (which will also be the width of the field needed for the counter value encoded in the seed row).
2. Initialize the width of compressed glue representations $l_g = 1$.
3. These will yield the width of padding $l_p = l_{pg} - l_g$ (which is a field encoded in the seed row).
4. Using l_{pg}, generate the functional tile set T_F (i.e. the tiles that perform phases 1 through 7 of the quine construction, following the descriptions previously provided for those phases). Note that l_{pg} will slightly impact the size of T_F by determining how many tiles must be hard-coded for the set of glue lookup entries. This is because for each symbol x in the constant-sized set of symbols that can be northern glue labels for the seed row tiles, there is a set of hard-coded tiles to represent $\texttt{PAD\_BIN}(x, l_{pg})$, which is as long as l_{pg}.
5. Get the count of unique horizontal glue labels, $g_h = |G_H|$, and unique vertical glue labels, $g_v = |G_V|$, in T_F.
6. If $l_g < \log(g_h)$ or $l_g < \log(g_v)$:
   (a) Set l_g equal to the maximum of $\log(g_h)$ and $\log(g_v)$.
   (b) If $l_{pg} < l_g$, set $l_{pg} = l_g$.
   (c) Return to Step 3 so that the process is redone and a (currently) accurate value for l_g can be determined.
7. Eventually l_g (and l_{pg}) can accommodate both g_h and g_v. This must happen since each increase of l_g allows twice as many glues to be encoded but only causes a constant number of additional tiles to be generated, namely 13. That is, for the encoding of each of the 13 possible northern glue symbols in the seed row tiles, one extra bit is added and thus one extra tile is needed for each.
8. Using the current l_{pg}, l_g, $l_p = l_{pg} - l_g$, and T_F, generate the tile set for the seed row, T_S, which includes the counter field padded to width l_{pg}, the encodings of the northern and eastern glues of the seed tile (both of length l_{pg}), the glue padding field of length l_p, and the glue lookup table containing the compressed glue lookup entries for the tiles of T_F (whose glue representations depend upon l_g).
9. Let $g'_h = |T_S| - 1$ (i.e. the number of unique horizontal glues between the tiles of the seed row). Note that the tiles of T_S add no new vertical glues since the tiles of T_F have tiles with glues that can bind to each of them.
10. If $l_{pg} < \log(g'_h + g_h)$ or $l_{pg} < \log(g_v)$, increase l_{pg} by 1, let $l_p = l_{pg} - l_g$ for this new value of l_{pg} and return to Step 3, since l_{pg} was not enough bits to encode all of the unique glue labels.
11. Eventually, l_{pg} will be large enough (this must happen since any increase only causes one additional tile for each of the counter field of width l_{pg} and the padding field of width l_p, so a constant number of added tiles and new glues, but a doubling of the number that can be represented.
12. Let the final tile set $Q = T_F \cup T_S$.

7 An Abstract Model of Self-assembly: Self-Describing Embedded Circuits

Sections 5 and 6 demonstrate that completely analyzing the behavior of an aTAM system, even when it is standard, involves a lot of details. This analysis is made harder by the asynchronous nature of the aTAM. It is necessary in the case of intrinsically universal systems (and thus of their quines) since intrinsic universality characterizes the behavior of the universal system. On the other hand, when one is merely interested in designing a standard system with a given set of productions, it is possible to use a so-called *self-descipting circuit* as a blueprint, which can then be compiled into a standard aTAM system. This device will be of use in section Section 10 to prove the existence of a standard aTAM system which strictly assembles the Sierpiński Cacarpet K^{∞}.

7.1 Embedded circuits As a way to design and analyze complex standard aTAM systems, these can be viewed as circuits drawn on $\mathbb{Z}^2$. When defined that way, one does not need to consider the effects of concurrency between different parts of the assembly, nor how the attachments are ordered. In exchange, a condition of *self-description* is needed for the circuit to translate into a TAS system. The following definitions can be skipped if one admits that a circuit is "like Figure 25", where wires coming into the same gate bend towards each other, so that the type of each gate g determines the set of inputs which will be received by its successors in addition to g's output (if any).

A circuit is made of gates. Each of these gates has a *function*, computing the outputs of the gates from the inputs. The gate is embedded on a unit square according to its *wiring*, which states where its inputs come from and where its outputs go to. The wiring and the function must be compatible: the wiring must have as many inputs as the function.

DEFINITION 7.1. (WIRE) *A wire is a pair of a list of distinct directions $\vec{D} \in \{N, E, S, W\}^*$ and a distinguished element $d \in D$. It is noted as a word on the alphabet $\{n, e, s, w\}$ containing the elements of D, with d capitalized. For instance, the wire $((N, E), E)$ is noted nE, while $((N, E), N)$ is noted Ne.*

Given a wire $w = (\vec{D}, d)$, its opposite $-w$ is $((-\vec{D}_i), -d)$, i.e. $-Ne = Sw$. The set of wires is noted Wires.

DEFINITION 7.2. (WIRING) *Given two integers i, o such that $i + o \leq 4$, a (i, o)-*wiring *w (i.e. a wiring with* in-degree *i and* out-degree *o) is given by*

- *a partition of $\{N, E, S, W\}$ into three sets, $w\cdot$*Inputs*, $w\cdot$*Outputs *and $w\cdot$*Inert*, where $w\cdot$*Inputs *is ordered*
- *a map $w\cdot$*output-num$: w\cdot$*Outputs$ \to \{0, \dots, o-1\}$;*
- *a map $w\cdot$*outwires$: w\cdot$*Outputs$ \to$* Wires *such that $-d$ is the distinguished element of $w\cdot$*outwires(d).

The set of all wirings is noted $\mathcal{W}$.

DEFINITION 7.3. (INPUT / OUTPUT / INERT SIDES, DEGREE) *Given a wiring $w \in \mathcal{W}$, the elements of $w\cdot$*Inputs *are the* input sides *of w, the elements of $w\cdot$*Outputs *are its* output sides*, and the elements of $w\cdot$*Inert *are its* inert sides*. The in-degree of w is the number of its input sides, its out-degree of w is its number of output sides.*

Additionally, each output side also determines the *input* sides on the following gate and their ordering. That is, a wiring w with $nE \in w\cdot$outwires can only have a wiring w' with $w'\cdot$Inputs $= (S, W)$ to its right.

The graphical representation of a wiring is given on Figure 24. Each direction in $w\cdot$Inputs has an incoming arrow, and each direction in $w\cdot$Outputs has an outgoing arrow. If $w\cdot$Inputs has only one element, the corresponding arrow is straight and simple. Likewise for the outgoing arrow when $w\cdot$outwires(d) is a singleton. When $w\cdot$outwires(d) is made of two adjacent directions, the corresponding arrows bend to come near each other; likewise, modulo rotation, when $w\cdot$outwires$(N) \in \{Ne, eN\}$, the corresponding arrow bends left, and right if it is Nw or wN. For pairs of opposite directions, an arrow with a double tip is used. Triple and quadruple inputs are not used in this paper. Inputs are numbered according to the order of $w\cdot$Inputs and output wires are numbered according to $w\cdot$output-num.

DEFINITION 7.4. (GATE) *Given an alphabet Σ two integers i, o, a gate g is a pair $\{$*func$: f$, *wiring$: w\}$, with f a function $f : \Sigma^i \to \Sigma^o$ and w an (i, o)-wiring.*

The set of gates on alphabet Σ is noted Gates(Σ).

These gates may be placed onto a subset of the grid $\mathbb{Z}^2$ to make *circuits*. An example of a circuit, a multiplier built from a handful of adders and auxiliary gates is given on Figure 25.

DEFINITION 7.5. (CIRCUIT) *Let Σ be an alphabet. Let D be an oriented subgraph of $\mathbb{Z}^2$, $\vec{I}$ be a sequence of arcs from $\mathbb{Z}^2 \setminus D$ to D and $\vec{O}$ a sequence of arcs from D to $\mathbb{Z}^2 \setminus D$.*

A circuit C with dependency graph D, input bus $\vec{I}$ and output bus $\vec{0}$ is a map $D \to$ Gates(Σ) *such that:*

- *for any arc E in direction $d \in \{N, E, S, W\}$ between two positions $a, b \in D$, let $w_a = C(a)\cdot$*wiring *and $w_b = C(b)\cdot$*wiring*; then we have $d \in w_a\cdot$*Outputs*, $-d \in w_b\cdot$*Inputs*, and $w_a\cdot$*Outputs$(d) = (-d, w_b\cdot$*Inputs$)$,*

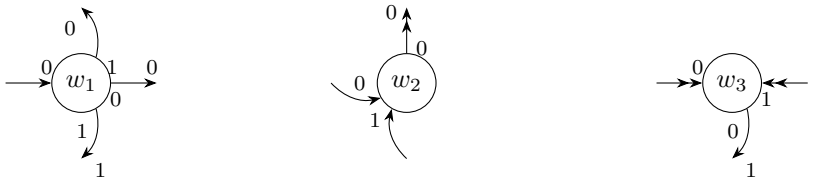


Figure 24: The graphical representation of three wirings: $w_1 = \{\text{Inputs} = (W), \text{Outputs} = \{N, S, E\}, \text{output-num} = \{N \to 1, E \to 0, S \to 1\}, \text{outwires} = \{Se, W, wN\}\}$, $w_2 = \{\text{Inputs} = (W, S), \text{Outputs} = \{N\}, \text{output-num} = \{N \to 0\}, \text{outwires} = \{Sn\}\}$ and $w_3 = \{\text{Inputs} = (W, E), \text{Outputs} = \{S\}, \text{output-num} = \{S \to 0\}, \text{outwires} = \{wN\}\}$. Neighboring input wires bend to come together into the wiring. Double arrows indicate a pair of opposite input arrows. The output wire in direction d bends to mirror $w{\cdot}\,\text{outwires}(d)$.

- *for any adjacent positions a, b in $\mathbb{Z}^2$ with no arc between a and b in D, $d \in C(a){\cdot}\,\text{wiring}{\cdot}\,\text{Inert}$ and $-d \in C(b){\cdot}\,\text{wiring}{\cdot}\,\text{Inert}$*
- *and for any arc e of $\mathbb{Z}^2$ in direction $d \in \{N, E, S, W\}$ between two positions $a \in D$ and $b \in \mathbb{Z}^2 \setminus D$, either:*
  - *$d \in C(a){\cdot}\,\text{wiring}{\cdot}\,\text{Inert}$ and e is neither an element of $\vec{I}$ nor of $\vec{O}$,*
  - *$d \in C(a){\cdot}\,\text{wiring}{\cdot}\,\text{Outputs}$ and e is an element of $\vec{O}$,*
  - *$-d \in C(a){\cdot}\,\text{wiring}{\cdot}\,\text{Inputs}$ and $-e$ is an element of $\vec{I}$.*

  The support of C is the vertex-set of D.

Let C be a circuit, and A the arc-set of its dependency graph D. A function $v : A \mapsto \Sigma$ *conforms* to C at a position $\vec{z}$ in the support of C, if for any $d \in C(\vec{z}){\cdot}\,\text{wiring}{\cdot}\,\text{Outputs}$, $v(o) = (C(v){\cdot}\,\text{func})(v(i_0), \ldots, v(i_m))_k$ with $i_0, \ldots, i_m$ the input arcs of $C(\vec{z})$, o its output arc in direction d and $k = C(\vec{z}){\cdot}\,\text{wiring}{\cdot}\,\text{output-num}(d)$.

A circuit is *well-founded* when its dependency graph has no cycle or infinite backwards paths. For a well-founded circuit, given a vector of inputs $\vec{\imath}$ indexed by $\vec{I}$, there is a *unique* function $\widetilde{C}(\vec{\imath}, -)$ which conforms to C and such that the values on $\vec{I}$ are $\vec{\imath}$. The circuit function $\bar{C} : \Sigma^{\vec{I}} \to \Sigma^{\vec{O}}$ is defined by $\bar{C}(\vec{\imath}) = (\widetilde{C}(\vec{\imath}, O_k))_k$.

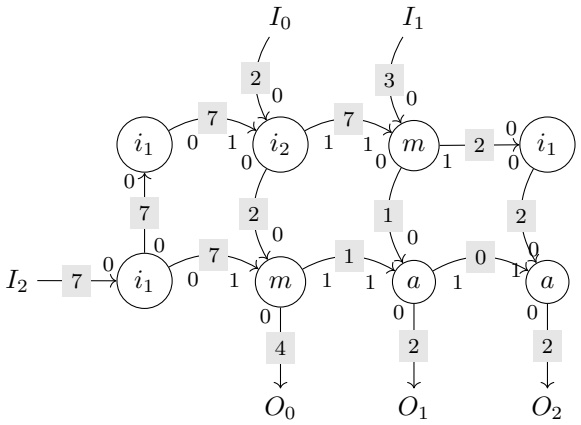
A circuit is *finitely rooted* when its input bus is finite and its number of gates with in-degree 0 is finite. It is *evaluable* if it is well-founded and finitely rooted.

The circuit M_2^1 on figure 25 is a multiplier. Take two natural integers $a < 10, b < 100$, pose $b = b_0 + 10b_1$ with $\forall i, 0 \leq b_i < 10$. Let $\vec{\imath}_{a,b} = (a_0, b_0, b_1)$. Then $\bar{C}(\vec{\imath}_{a,b}) = \vec{o}$, with $ab = \sum o_i 10^i$. Figure 25 gives an example of the evaluation of the circuit on inputs 32 and 7; the values of $\widetilde{C}(\vec{\imath}, -)$ on each arc are given on the figure.

When the alphabet Σ is structured in two layers, i.e. $\Sigma = A \times B$, the function f of a gate g may act independently on layer A and layer B, i.e. $f = f_A \otimes f_B$. Then, taking $g_A = \{\text{wiring} : w, \text{func} : f_A\} \in \text{Gates}(A)$ and $g_B = \{\text{wiring} : w, \text{func} : f_B\} \in \text{Gates}(B)$, g can be written as $g_A \otimes g_B$. In other words, the tensor product $\otimes$ applies to pairs of gates *with the same wiring*.

7.2 Self-description The process of self-assembly in the aTAM and the evaluation of an evaluable embedded circuit are somewhat alike, in that information propagates from an initial region outwards. In both processes, there is no global synchronisation, but each step can take place as soon as its inputs are ready. In the aTAM, the initial region is the seed, while in a circuit, it is the input bus together with the gates of in-degree 0.

There are three differences between these processes. First, the input bus of a circuit does not have an analog in the aTAM. Hence, aTAM systems are akin to *closed* circuits. Secondly, there can be competition between attachments in the aTAM, as well as mismatches, while in circuits the input and output directions of each gate are fixed. This entails that an aTAM derived from a circuit by the compilation process detailed below will be standard. The last and most important difference is that in a circuit, the outputs of each gate depends not only on its input, but also on the function of the gate. For an aTAM system to simulate a circuit, this information needs to be derived from the inputs of the gate. Self-description captures the possibility to do so.



$$\begin{cases} i_1 : \Sigma \to \Sigma \\ i_1(x) = x \\ i_2 : \Sigma^2 \to \Sigma^2 \\ i_2(x,y) = (x,y) \end{cases} \qquad \begin{cases} m : \Sigma^2 \to \Sigma^2 \\ m(x,y) = (xy \bmod 10, \lfloor xy/10 \rfloor) \\ a : \Sigma^2 \to \Sigma^2 \\ a(x,y) = ((x+y) \bmod 10, \lfloor (x+y)/10 \rfloor) \end{cases}$$

Figure 25: A multiplier circuit M_2^1 and the definition of the functions of its gates. The alphabet is the set of digits $\mathbb{Z}/10\mathbb{Z}$. The values in the grey squares are an example of evaluation: $32 \times 7 = 224$

Definition 7.6. (Self-describing circuit) *An evaluable circuit C on alphabet Σ is* self-describing *on input vector $\vec{\imath}$ if there is a decoding function* dec-gate : $(\Sigma \times \{N,E,S,W\})^{<4} \to \text{Gates}(\Sigma)$ *such that for any position p with incoming arcs $(e_0, \dots, e_{k-1}) = p + (C(p) \cdot \text{wiring} \cdot \text{Inputs})$:*

$$\text{dec-gate}((\widetilde{C}(\vec{\imath}, e_0), d_0), \dots, (\widetilde{C}(\vec{\imath}, e_{k-1}), d_{k-1})) = C(p),$$

where d_j is the direction of e_j.

A closed self-describing circuit really is a circuit with only one function of each in-degree. Having one function per in-degree is a harmless technicality: in a circuit, successive gates have compatible wirings, so the in-degree of each gate is known from the wiring of any of its predecessor gates, whether or not the circuit is self-describing.

Lemma 7.1. *Let C be a closed self-describing circuit. There is a closed circuit C_1 with only only one function of each in-degree such that $\widetilde{C} = \widetilde{C_1}$.*

Proof. C_1 has the same wirings as C, and for each i, all of its gates of in-degree i have as function $f_i : x_0 \dots x_{i-1} \mapsto (\delta_C(x_0, \dots x_{i-1}) \cdot \text{func})(x_0, \dots x_{i-1})$, where δ_C is the decoding function of C. ∎

7.3 Locally deterministic patterns If C is a self-describing closed circuit, then from $\widetilde{C}$, one gets a coloring of the arcs of its dependency graph with the property of *local determinism*.

Definition 7.7. (Locally deterministic arc coloring) *Let P be a coloring of the arcs of some acyclic oriented subgraph G of $\mathbb{Z}^2$. For each vertex $v \in G$ with in-degree δ, the input vector of v is $\vec{\imath}_v = ((d_0, P(e_0)), \dots, (d_{\delta-1}, P(e_{\delta-1})))$, where e_i is the i-th incoming edge of v and d_i its direction*

P is locally deterministic if there are two prediction functions pred-inputs : $\{N,E,S,W\} \times \Sigma \to \mathcal{P}(\{N,E,S,W\})$ *and* pred-symbols : $(\{N,E,S,W\} \times \Sigma)^{<4} \times \{N,E,S,W\} \to \Sigma$ *such that for each arc e from vertex a to vertex b in direction $d \in \{N,E,S,W\}$,*

- $P(e) = \text{pred-symbols}(\vec{\imath}_a, d)$
- $\text{pred-inputs}(d, P(e))$ *is the set of directions of the incoming arcs of b,*

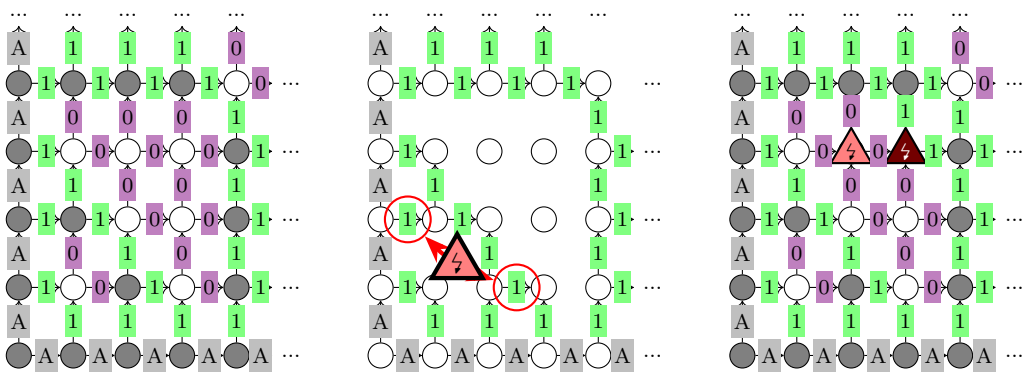


Figure 26: Three arc colorings on the alphabet $\Sigma = \{A, 0, 1\}$. All arcs go either to the right or up. The coloring C_1, in the upper left is locally deterministic. In the upper-right, C_2 is not because the two circled arcs have value 1 and direction E, but the vertices they point into have different incoming directions ($\{E\}$ versus $\{N, E\}$). At the bottom, C_3 is not locally deterministic, because the two triangle vertices get the same incoming values, but have different outputs: $(0, 0)$ versus $(1, 1)$.

LEMMA 7.2. *Let C be a closed self-describing circuit on alphabet Σ, with dependency graph $D_C = (V, A)$. Let C' be the self-describing circuit on alphabet $\Sigma \times \mathcal{W}$ where each gate outputs its wiring in addition to its normal output. Then the function $\widetilde{C'}$ is a locally deterministic coloring of D_C.*

Proof. A candidate function for pred-inputs takes as input the direction d of an arc $e : a \to b$, and the value of $\widetilde{C'}(e)$, that is, an element of Σ corresponding to $\widetilde{C}(e)$ as well as the wiring w of the gate at a. Thus, the function $(d, x, w) \mapsto w \cdot \text{outwires}(d)$ must, by the constraints on neighboring gates in a circuit, output the set of directions of the incoming arcs at b.

Likewise, a candidate function for pred-symbols takes as input the direction of the arc $e : a \to b$, and a vector $\vec{m} \otimes \vec{d}$ of the inputs coming into a with their directions. Reorder $\vec{m} \otimes \vec{d}$ so that $\vec{d}$ is in the order of the inputs of $C(a)$. Let $\{\text{wiring} : w, \text{func} : f\} = \text{dec-gate}(\vec{m} \otimes \vec{d})$. Then taking pred-symbols$(e) = (f(\vec{m}), w)$ works. ∎

From locally deterministic patterns to aTAM systems

DEFINITION 7.8. (VERTEX TYPE, ATLAS) *Let P be a coloring of the arcs of some acyclic oriented subgraph G of $\mathbb{Z}^2$. For a vertex v of G, its* vertex type *is the partial function $\bar{v} : d \in \{N, E, S, W\} \mapsto (o, x)$, where $o = -d$ if the edge e in direction d from v is an incoming edge, $o = d$ if it is an outgoing edge, and $x \in \Sigma$ is $P(e)$. If v has no edge in direction d, then $\bar{v}(d)$ is undefined.*

The atlas *of P is the set of its vertex types.*

A locally deterministic coloring can be reconstructed from its atlas and the positions of its in-degree 0 vertices.

LEMMA 7.3. *Let P_1, P_2 be locally deterministic colorings of the arcs of some oriented subgraphs of $\mathbb{Z}^2$, G_1 and G_2 respectively. If*

1. *G_1 and G_2 are acyclic and have no infinite backwards paths,*
2. *P_1 and P_2 have the same atlas,*
3. *G_1 and G_2 have the same vertices with in-degree 0,*

then $G_1 = G_2$ and $P_1 = P_2$.

Proof. By induction on the longest path to each position in G_1. ∎

Finally, a locally deterministic pattern with a unique in-degree 0 vertex can be self-assembled in the aTAM.

LEMMA 7.4. *Let Σ be some finite alphabet, and let P be a locally deterministic coloring of the arcs of some acyclic graph G with no infinite backwards path. Assume the maximal in-degree of a vertex in G is δ and that G has only one vertex with in-degree* 0.

Then there is an aTAM system S_P with temperature δ with P as its only final production.

Proof. Let A_P be the atlas of P, pred-inputs and pred-symbols the prediction functions of P. By convention, suppose pred-inputs returns an input vectors which is ordered clockwise, starting from direction N.

First define the glues of S_p and their strength. Each glue is a pair $(s, d) \in \Sigma \times \{N, E, S, W\}$. For any symbol $s \in \Sigma$ and direction $d \in \{N, E, S, W\}$, let $\vec{\imath} =$ pred-inputs(s, d). If d is the first element of $\vec{\imath}$, then its strength is $\delta + 1 - |\vec{\imath}|$, otherwise it is 1. Hence, for any vertex type $v \in A_P$ with input vector $\vec{\imath}_v$, the sum of the strengths on the input sides is δ.

Then define the set of tile types S_p to be A_p, with the glue on the d side of the tile type $\bar{v}$ being $\bar{v}(d)$ if defined, and the null glue otherwise. The seed tile of S_p is the only vertex type of A_p with in-degree 0.

Indeed, any production of S_P is an initial subset of P, as can be seen by induction on the attachments.

Then, consider a production p of S_P. If G contains some position not covered by p, then because G contains neither loops nor infinite paths, it must contain some v with all its predecessors within dom(p). But then the total strength of glues into v is δ, and the tile corresponding to $P(v)$ must be attachable there. Hence p is not terminal.

Finally, S_P assembles P. ∎

Putting all this together, it is possible to compile a self-describing circuit into a self-assembling system.

THEOREM 7.1. *Let C be a self-describing evaluable circuit in which exactly one gate has no inputs. Then there is a standard aTAM system S_C which strictly self-assembles* dom(C).

Proof. By Lemmas 7.2 and 7.4. ∎

8 A Geometric limitation of aTAM computation: The Tree Pump Theorem

The dynamics of the aTAM are limited by pumping principles in the style of [29], whereby under some conditions, their behavior must become periodic. The one presented here deals with skinny productions, that is those who do not encircle any square larger than N for some fixed N. The crucial property which makes them simple(-ish) is their bounded treewidth [8]. One remarkable feature of this pumping theorem is that it allows pumping *in a chosen direction*. In Section 11, this pumping principle will be the final argument for ruling out the least connected of shapes from generating DSSFs which can be self-assembled.

THEOREM 8.1. (TREE PUMP) *For any aTAM system $\mathcal{T} = (S, \sigma, \tau)$ with σ finite and connected, define the following sets of assemblies:*

- *for any integer m, $C_m[\mathcal{T}]$ is the set of assemblies of $\mathcal{T}$ which encircle an $m \times m$ square;*
- *for any real k and vector $\vec{d}$, $B_{k,\vec{d}}[\mathcal{T}]$ is the set of assemblies of $\mathcal{T}$ which do not cover any position $\vec{p}$ such that $\vec{p} \cdot \vec{d} > k + |\text{dom } \sigma|$*
- *for any vector $\vec{d}$, $P_{\vec{d}}[\mathcal{T}]$ is the set of* ultimately periodic *assemblies A of $\mathcal{T}$ such that there is a vector $\vec{p}$ with $\vec{p} \cdot \vec{d} > 0$ and a non-empty sub-assembly $a \sqsubseteq A$ such that $a + \vec{p} \sqsubseteq a$.*

Then, there is a function $F : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that for aTAM system $\mathcal{T}$ with n tile types and a 1-tile seed, integer m and unit vector $\vec{d}$ of $\mathbb{R}^2$,

$$\mathcal{A}_{\square}[\mathcal{T}] \cap (C_m[\mathcal{T}] \cup B_{F(n,m),\vec{d}}[\mathcal{T}] \cup P_{\vec{d}}[\mathcal{T}]) \neq \varnothing.$$

The Tree Pump Theorem states that in order to do a meaningful amount of computation, a self-assembling system with n tiles needs to encircle large squares —say, of size m, thus hitting $C_m[\mathcal{T}]$. If it does not, then its productions look very much like trees drawn on $\mathbb{Z}^2$ with a m-cell wide brush. A branch of that tree then behaves like a finite automaton. If that automaton stops, the assembly goes no further than $F(n, m)$ in any given direction $\vec{d}$, which gives a final production in $B_{F(n,m),\vec{d}}[\mathcal{T}]$. If not, these long branches must have an ultimately periodic behavior which gives a final production in $P_{\vec{d}}[\mathcal{T}]$.

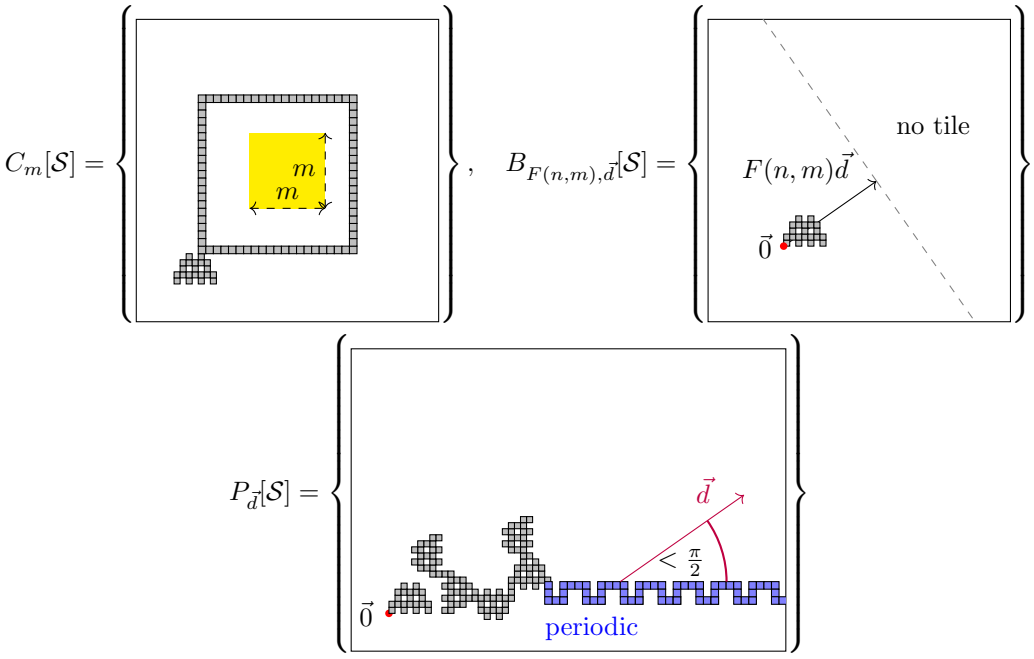


Figure 27: The three sets defined by theorem 3.4: $C_m[\mathcal{S}]$ is the assemblies which encircle an $m \times m$ square, $B_{F(n,m),\vec{d}}[\mathcal{S}]$ is the set of assemblies which do not reach further than $F(n,m)$ in direction $\vec{d}$, and $P_{\vec{d}}[\mathcal{S}]$ is the assemblies which contain a periodic path with period $\vec{p}$ such that $\vec{p} \cdot \vec{d} > 0$.

The proof of theorem 3.4 needs some ancillary definitions. The objects they introduce may seem overly sophisticated in view of the concreteness of the statement of theorem 3.4, but one needs to soldier on. The reader may find solace in knowing that they are the result of persistent failure in one of the author's search of a more down-to-earth argument. The proof plainly needs systems endowed with the ability to escape the confines of the Euclidian plane $\mathbb{Z}^2$ and those of a merely infinite time. Hence, the definition and use of *Ordinal-length* sequences of *Free Assemblies*.

8.1 Free Assemblies In this appendix, assemblies sit on a graph called an *assembly support* instead of $\mathbb{Z}^2$.

Definition 8.1. (Assembly support) *An* assembly support *is a directed graph G with labels in $\{N, E, S, W\}$ on its arcs, such that:*

- *each vertex has at most one incoming arc with each label,*
- *each vertex has at most one outgoing arc with each label,*
- *for any cycle c of G, the sum of the labels of the arcs of c is equal to $\vec{0}$ as a vector of $\mathbb{Z}^2$,*
- *for any path π of 1 or 3 arcs from u to v, if the sum in $\mathbb{Z}^2$ of the labels of π is $\vec{d} \in \{N, E, S, W\}$, then there is an arc from v to u with label $-\vec{d}$.*

An example of a correct assembly support is given on Figure 29, while Figure 28 shows some counter-examples of labeled graphs which are not assembly supports. Because of the conditions on the incoming and outgoing arcs of each vertex, it makes sense to represent the vertices as tiles, and arcs as sides the tiles share. This hopefully helps build the intuition that assembly support are "like $\mathbb{Z}^2$, but long separated paths which should come to the same position may avoid each other". On Figure 28, the direction corresponding to each side of the tile is given explicitely. With correct Assembly Supports, like the ones on Figure 29, the convention is that the direction

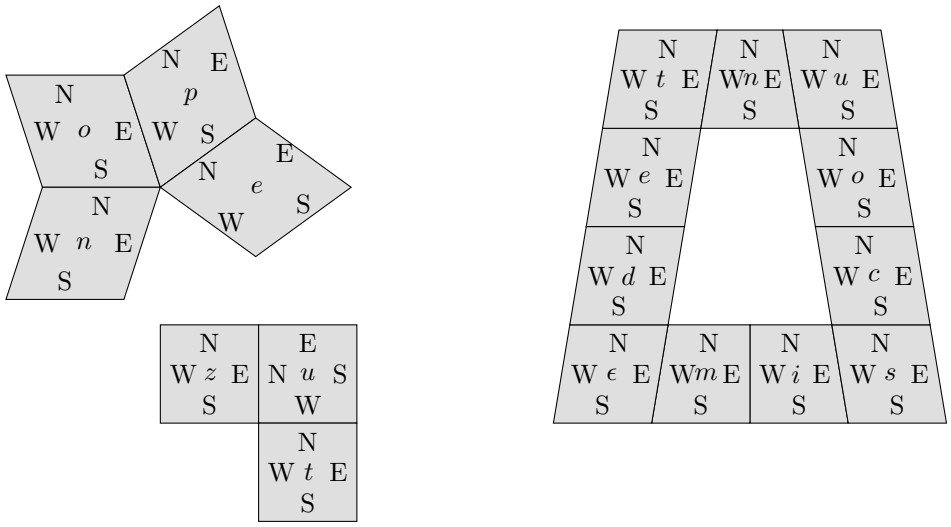


Figure 28: *Ceci n'est pas un Assembly Support*: three labelled graphs which fail to be assembly supports. Each vertex is represented by a tile, the labels of its sides are the labels of its outgoing arcs. A label on a side with no neighbor corresponds to an exterior arc. The bottom-left graph has a path of length 1 with label E from z to u but the $-E = W$ neighbor of u is t. Also, the N neighbor of t is u, which has no S neighbor. The one in the top right has a 3 arc path "*nope*" whose labels sum to $N + E + S = E$, but n is not the W neighbor of e (actually, there is none). The one on the right has a cycle "$\epsilon\, miscounted$" for which the sum of the labels is $3E + 3N + 2W + 3S = E$, which is not $\vec{0}$.

of each side corresponds roughly to its direction on the page, and small adjustments allow tiles which would otherwise be adjacent on the page not to be. The appearance of tiles avoiding each other by going in the third dimension is deliberate, as a way to build intuition.

An *embedding* from an assembly support G to another assembly support G' is a graph embedding which preserves the label of the arcs. An *isomorphism* between G and G' is a graph isomorphism preserving the arc labels.

Definition 8.2. (Perspective difference, Translation) *For any two vertices (z, z') of a connected component of an assembly support G, their* perspective difference $z' \mathring{-} z$ *is $\sum_{a \in p} a$, where $p \in \{N, E, S, W\}^*$ is the sequence of labels of a path from z to z'. For any embedding $e : G \to \mathbb{Z}^2$, $e(z') - e(z) = z' \mathring{-} z$. By convention, z, z' are in different connected components, $z' \mathring{-} z = \infty$.*

A translation between two subgraphs $A \subset G$ and $B \subset G'$ of two assembly support is an isomorphism from A to B which preserves perspective differences. Any isomorphism between connected subgraphs is a translation.

For any Assembly Support G and $z \in G$, $\text{squash}_{z,G} : G \to \mathbb{Z}^2$ is the embedding $z' \mapsto (z' \mathring{-} z)$. Since changing z in $\text{squash}_{z,G}$ only amounts to a translation, it will generally be omitted. The index G will be omitted as well when this causes no ambiguity.

The analysis in the Tree Pump Theorem relies on an examination of the *holes* of the productions of $\mathcal{T}$. These holes need to be suitably defined now that the assemblies are no longer necessarily planar.

Definition 8.3. (Exterior Arc) *An* exterior arc *of an assembly support G is a pair of a vertex v of G and a direction d such that v has no arc labelled d in G.*

Definition 8.4. (Converging Exterior Arcs, Growth) *Two exterior arcs (v, d), (v', d') are* converging *when $v' \mathring{-} v = d' - d$ –they virtually point to the same empty position.*

Given an assembly support G and a set Z of converging arcs, the assembly support $G + Z$ (G grown by Z) is $G \cup \{\zeta\}$, where there is an arc $z \to \zeta$ (for $z \in G$) in direction d whenever $(z, d) \in Z$.

Definition 8.5. (Step, Exterior Path, Hole) *There is a* step *between two arcs or exterior arcs (v, d) and (v', d') if:*

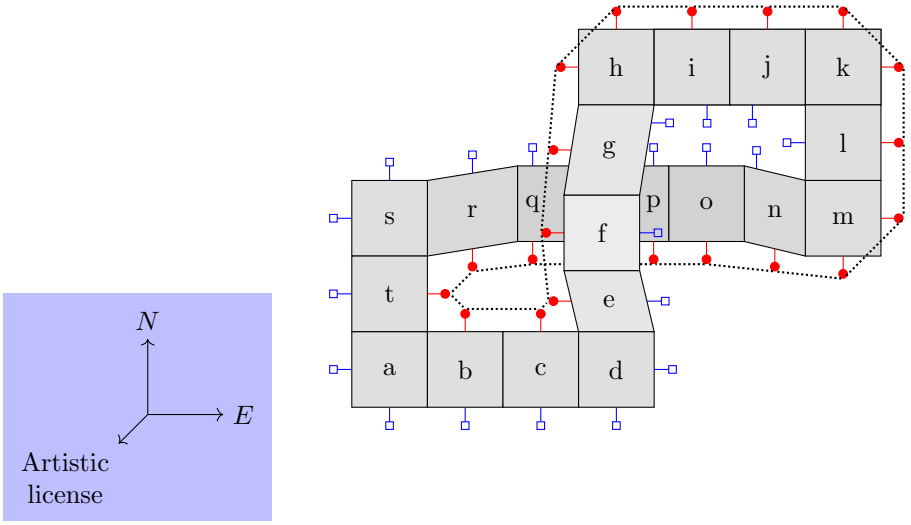


Figure 29: An example of an assembly support, the shifts in the 3rd dimension between e and g as well as between r and n are artistic license to show the absence of adjacencies between those two branches. The sum of the vectors between neighbors on the cycle $abcdefghijklmnopqrsta$ is $3E+4N+3E+2S+6E+2S=\vec{0}$. The sum of vectors between neighbors on the path $fedcbatsrqp$ is $2S+3W+2N+3E=\vec{0}$, yet the vertices f and p are distinct. The small pending vertices next to the tiles are the exterior arcs; they form two holes, one for the square exterior arcs, and one for the circles. The dotted lines shows the steps between the exterior arcs of the "circle" hole. The step between the exterior arcs (m,S) and (m,E) stems from the first case of the definition, the one between (b,N) and (c,N) from the second case, and the one between (t,E) and (r,S) because of the third. In this last case, s acts as the pivot between r and s. Both holes have perimeter 20.

- *$G=G'$ and d,d' form a $\pm\frac{\pi}{2}$ angle,*
- *$d=d'$ and there is an (non-exterior) arc from v to v' with label d'', with d,d'' forming a $\pm\frac{\pi}{2}$ angle*
- *d,d' form a $\pm\frac{\pi}{2}$ angle and there is a cell v'' with $v''+d=v$ and $v''+d'=v'$.*

A (simple) exterior path *is a sequence $P=(p_0,p_1,\dots,p_k)$ of arcs, such that:*

- *for each i, there is a step between p_i and p_{i+1}, and*
- *for $0<i<k$, p_i is an exterior arc.*

A hole *is a simple exterior path which loops back to its starting exterior arc.*
The perimeter *of a hole $H=(h_0,\dots h_k=h_0)$ is the set of vertices appearing in the exterior vertices h_i.*

These definitions are illustrated on Figure 29, which features an assembly support with two holes of perimeter 20. These holes are "morally infinite", so they resist attempts at a definition of area —at least the author's naive ones. For an assembly which embeds into $\mathbb{Z}^2$, the free definition of hole corresponds to the intuition of a hole in an assembly, up to the detail that the exterior of the assembly forms a hole, so that all finite assemblies have at least one hole. Note that exterior paths are oriented: they turn in the positive direction around each tile.

Definition 8.6. (Free Assembly) *A free assembly A of the TAS $\mathcal{T}=(T,\sigma,\tau)$ is composed of an assembly support $A\cdot$support and a total function $A\cdot\text{tile}:A\cdot\text{support}\to T$.*
For a TAS $\mathcal{T}$, the set of free assemblies of $\mathcal{T}$ is written $\mathcal{T}^{\text{Free}}$.

When there is an injective embedding $e:A\cdot\text{support}\to G$, it naturally defines a free assembly $e(A)$ with $e(A)\cdot\text{support}=e(A\cdot\text{support})$ and for $z\in e(A\cdot\text{support})$, $e(A)(e(z))=A(z)$. An assembly A *embeds* into $\mathbb{Z}^2$ if squash is a one-to-one embedding $A\cdot\text{support}\to\mathbb{Z}^2$; as above, this embedding naturally defines an assembly $\text{squash}(A)$.

Definition 8.7. (Free Attachment, Free Assembly Sequence) *Let A, A' be free assemblies of some TAS system $\mathcal{T} = (T, \sigma, s)$, and $\eta : A \to A'$ an embedding. The tuple (A, A', η) is an assembly candidate if:*

- *there is a $z \in A'\cdot\text{support}$ such that $\eta : A\cdot\text{support} \to A'\cdot\text{support}\setminus\{z\}$ is a translation,*
- *for any $z' \in A\cdot\text{support}$, $A\cdot\text{tile}(z') = A\cdot\text{tile}(\eta(z'))$.*

The definitions of attachment, assembly sequence, production and terminal production extend to free assemblies. The $\mathbb{Z}^2$ version of each definition is simply the particular case where every assembly embeds into $\mathbb{Z}^2$.

For a TAS $\mathcal{T}$, the set of free assembly sequences of $\mathcal{T}$ is written $\mathcal{H}^{\text{Free}}[\mathcal{T}]$.

The definition of free attachment is illustrated on Figure 30. The fact that η is an isomorohpism ensures that parts of the assembly which do not touch the position z are unaffected by the attachment. This preserves the locality of the aTAM process, in contrast to what happens in the FTAM [12] for instance. Hence, while the above definition of attachment is given from the point of view of A', "after the fact", from the point of view of A, an attachment candidate can be defined from a tile type t and a set Z of convergent exterior arcs: $t@Z$ is the attachment candidate (A, A', η) where:

- $A'\cdot\text{support} = A\cdot\text{support} + Z$, and ζ is the vertex of $A'\cdot\text{support}$ which is not in $A\cdot\text{support}$,
- η is the identity on $A\cdot\text{support}$,
- $A'\cdot\text{tile}(\zeta) = t$,
- $A'\cdot\text{tile}(z) = A\cdot\text{tile}(z)$ whenever $z \neq \zeta$.

For clarity, a "normal" assembly will be referred to as a $\mathbb{Z}^2$-assembly, likewise for the other concepts which were just endowed with a "free" variant.

Embeddings act not only on assembly supports and assemblies, but also on assembly sequences. In doing so, they may break the correctness of attachments if they are not one-to-one; when that happens, the sequence is cut short.

Definition 8.8. (Assembly Sequence Embedding) *Let $\alpha = (A_0 \to A_1 \to \ldots) \in \mathcal{H}^{\text{Free}}[\mathcal{T}]$, let $G = (\lim\alpha)\cdot\text{support}$ and G' an assembly support. Let $e : G \to G'$ be an embedding; since up to translation, $A_0\cdot\text{support} \subset A_1\cdot\text{support} \subset \ldots \subset (\lim\alpha)\cdot\text{support}$, for each i, e induces an embedding from $A_i\cdot\text{support}$ into G'.*

Let k be the largest i such that e is an isomorphism on $A_i\cdot\text{support}$, then $e(\alpha)$ is the assembly sequence $(e(A_0), \ldots, e(A_k))$.

8.2 Ordinal Assembly Sequences It is often practical to consider what happens in an assembly sequence α after it has placed an infinite number of tiles. For this, *ordinal* assembly sequences become useful.

Definition 8.9. (Ordinal Assembly Sequence) *Let $o \in \omega_1$ be a countable ordinal, an o-Assembly Sequence starting from an assembly A_0 is a sequence of length o of free assemblies such that for any $i < j \leq o$, there is an injective embedding $\eta_{i\to j} : A_i \to A_j$ with:*

- *for $i < o$, $(A_i, A_{i+1}, \eta_{i\to i+1})$ is a free attachment,*
- *for $i < j < k \leq o$, $\eta_{j\to k} \circ \eta_{i\to j} = \eta_{i\to k}$*

The i-th production of α is A_i, and its i-th attachment is $\eta_{i\to i+1}$.

For an o-Assembly Sequence α, the notation $\lim\alpha = A_o$ is consistent with the case of usual Assembly Sequences, i.e. ω-Assembly Sequences.

These sequences correspond to the intuitive generalization of attachment sequences to "times larger than infinity". In particular, any tile attached through an ordinal attachment sequence reaches the starting assembly of the sequence through a finite number of attachments. Indeed, for a tile to have been added through an ordinal

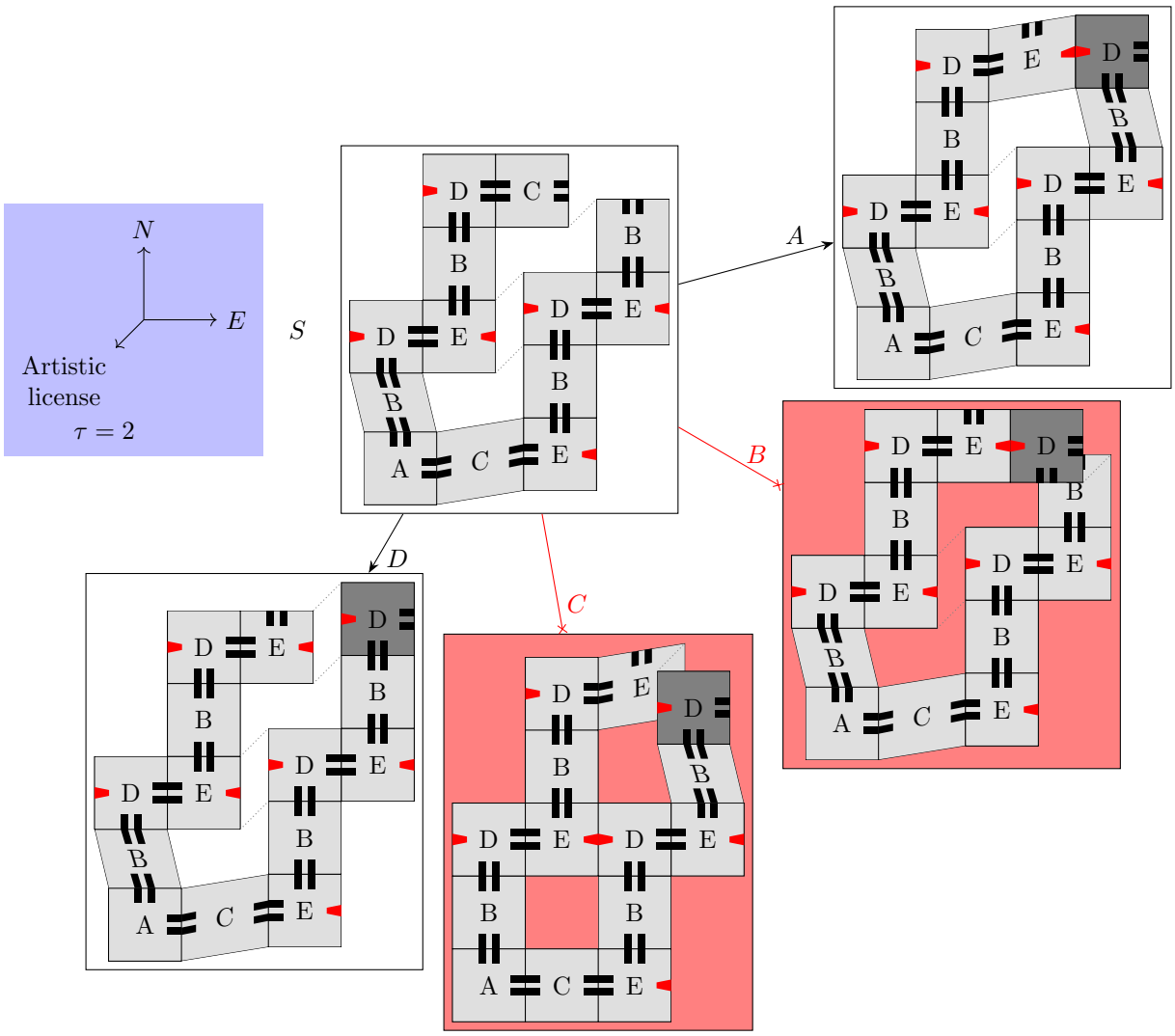


Figure 30: Some possible and impossible free attachments from a starting free assembly S, at temperature τ. Again, the third coordinate is a visual aid to show non-adjacencies; small dashed lines link points which embed in the same position in $\mathbb{Z}^2$. Assemblies A and D are reachable through one attachment $t_D@z$, where in each case, z is the position in dark gray. The assembly B is an attachment candidate, but it is not stable, since the new tile t_D only binds through a strength-1 glue. The assembly C can not even form an attachment candidate from S, as $C\cdot$ support does not contain an isomorphic copy of $S\cdot$ support

assembly sequence α at time t, the neighbors to which it attaches must have been attached at some time $t' < t$. Since these times are ordinals, such an decreasing sequence of times reaches 0 in a finite number of steps. In particular, these ordinal assembly sequences produce the same productions as the usual ω-assembly sequences.

This remark formalizes thus:

REMARK 1. *Let $\mathcal{T}$ be an assembly system, o a countable ordinal, α an o-Assembly Sequence, $k < o$ and $t@Z$ the k-th attachment of α. There is a sequence α' of length o' and a bijection $\iota : o \to o'$ such that:*

- *for all $t < o'$, the attachments α_t and $\alpha'_{\iota(t)}$ are the same,*
- *$\iota(k)$ is finite.*

LEMMA 8.1. *Let $\mathcal{T}$ be an assembly system, o a countable ordinal, and α an o-Assembly Sequence. Then if o is infinite, there is a ω-Assembly Sequence α' such that $\lim \alpha' = \lim \alpha$.*

Proof. The proof goes by transfinite induction.

If $o = \omega$, then α itself satisfies the conclusion of the lemma.

If $o = p + 1$ is a successor ordinal, by induction hypothesis, there is an ω-assembly sequence α'' such that $\lim \alpha'' = A_p$, where A_p is the p-th production of α. Let $t@Z$ be the p-th attachment of α; there is an integer k such that at time k, all the origin vertices of the arcs in Z are attached in α''. The attachment sequence $\alpha' = (\alpha''_0, \dots, \alpha''_k, t@Z, \alpha''_{k+1}, \dots)$ satisfies the conclusion of the lemma.

If o is a limit ordinal, there is a sequence $(\alpha'^i)_{i<o}$ of ω-assembly sequences such that for each i, $\lim \alpha'^i = A_i$. Let A'^i_j be the j-th production of α'^i. Since o is countable, it has cofinality ω; pick an increasing sequence $\epsilon : \omega \to o$ such that $\sup_{i<\omega} \epsilon(i) = o$. Let α' be the enumeration of the set $\{A'^{\epsilon(i)}_j | j \leq i < \omega\}$ ordered lexicographically according to (i, j). This sequence α' is an attachment sequence, and it satisfies $\lim \alpha' = \bigcup_{i<o}(A'^{\epsilon(i)}_i) = \lim \alpha$. ∎

8.3 Holes and Fizziness The Tree Pump Theorem is about tree-like assemblies. Since trees are acyclic graphs, the *holes* of the assemblies are inherently limited and thus play an important part in characterizing how these tree-like assemblies behave. Fizziness is the tendency of assembly sequences to create a lot of holes.

DEFINITION 8.10. (FIZZINESS) *Let A, A' be two free assemblies with $A\cdot\text{support} \subset A'\cdot\text{support}$, the* fizziness $\text{fz}(A, A')$ *is the number of holes of A' whose perimeter is not contained in A.*

Let $\alpha = (A_0 \to A_1 \to \dots)$ be an Assembly Sequence in $\mathcal{H}^{\text{Free}}[\mathcal{T}]$. The fizziness of α, noted $\text{fz}(\alpha)$ is the sequence $i \mapsto \text{fz}(A_i, A_i + 1)$.

A sequence α is more fizzy than β, written $\alpha >_{\text{fz}} \beta$ if $\text{fz}(\alpha) > \text{fz}(\beta)$ lexicographically.

LEMMA 8.2. (MAXIMAL FIZZINESS) *Let $\mathcal{T} = (S, \sigma, \tau)$ be a seeded assembly system. Assume σ has a finite number of non-null glues on its external edges. Then there is an ω-Assembly Sequence $\alpha_{\max} \in \mathcal{H}^{\text{Free}}[\mathcal{T}]$ with maximal fizziness among ω-Assembly Sequences.*

Proof. Using Kőnig's Lemma. ∎

Note that this lemma does not hold for o-assembly sequences with $o > \omega$. Indeed, after time ω, there might be an infinity of possible attachments, thwarting Kőnig's Lemma.

LEMMA 8.3. (FIZZINESS-INCREASE OF EMBEDDINGS) *Let $\mathcal{T}$ be a seeded TAS, and $\alpha \in \mathcal{H}^{\text{Free}}[\mathcal{T}]$. Let $G = \text{support}\,(\lim \alpha)$, G' an assembly support, and $e : G \to G'$ an embedding.*

Then:

1. *$e(\alpha)$ is a free assembly sequence,*
2. *$e(\alpha) \geq_{\text{fz}} \alpha$*
3. *$e(\alpha) >_{\text{fz}} \alpha$ if e is not injective on $\text{dom}\,(\lim \alpha)$.*

Proof. Let $\alpha = (t_i @ z_i)_i$ be an assembly sequence in $\mathcal{H}^{\text{Free}}[\mathcal{T}]$.

For Item 1, the attachments of $e(\alpha)$ are valid by definition of k. They are stable since for each attachment, the edges adjacent to z_i which make this attachment stable are preserved by e.

For Item 2, for each assembly A of α which is mapped injectively, each hole of α is mapped by e to a hole of $e(A)$ with the same labels on its border.

For Item 3, if e is not injective on dom $(\lim \alpha)$, then there are $i < k$ such that $e(z_i) = e(z_k)$. Since α is a valid assembly sequence, $z_i \neq z_k$, there are two different paths from the seed to $e(z_i)$. From these two paths, it is possible to construct a hole in dom $(e(\alpha))$ which is not a hole in dom (α). Hence, from the first attachment where this hole appears, the assemblies of $e(\alpha)$ are more fizzy than the corresponding ones in α. ∎

LEMMA 8.4. (MAXIMAL SEQUENCES ARE FLAT) *Let $\mathcal{T}$ be a seeded TAS, S an assembly of $\mathcal{T}$ and $X \subset \mathcal{H}^{\text{Free}}[\mathcal{T}, S]$ be a set of Free, Ordinal Assembly Sequences such that for any $\alpha \in X$, there is a $\alpha' \in X$ such that $\alpha' \geq_{\text{fz}} \text{squash}(\alpha)$.*

Assume α is a free assembly sequence with maximal fizziness within X. Then α embeds into $\mathbb{Z}^2$.

Proof. If α does not embed into $\mathbb{Z}^2$ by squash, then squash(α) is fizzier than α and thus α cannot be maximal in X. ∎

8.4 The Window Movie Lemma The Window Movie Lemma [29] applies to free assembly sequences as well as to $\mathbb{Z}^2$ assembly sequences. In order to account for holes, movies need to record not only the glues appearing on either side of the window, but also the paths between edges of the window through either side.

DEFINITION 8.11. (WINDOW, FRAME, MOVIE) *Let $\alpha = (A_0, A_1, \ldots)$ be an assembly sequence of $\mathcal{H}^{\text{Free}}[\mathcal{T}]$. A* window *$W$ of α is a cut of $(\lim \alpha) \cdot$ support separating it into two connected components* $\text{Near}(W)$ *and* $\text{Far}(W)$.

The width $w(W)$ of a window is the maximum perspective difference between two vertices along the window.

Let $\text{Arcs}(W)$ *be the set of arcs (either normal or external) of $(\lim \alpha) \cdot$ support through W.*

The frame *f on W at time k records for each arc $a = (v, d) \in \text{Arcs}(W)$:*

- *$f(a) \cdot$ present: whether $v \in A_k \cdot$ support, and if so,*
- *$f(a) \cdot \text{glue} = A_k \cdot \text{tile}(v)(d)$, as well as*
- *$f(a) \cdot$ successor the arc of* $\text{Arcs}(W)$ *which is reachable from a through an exterior path which does not cross W, if there is one.*

The movie *associated with W is the ordered set $\mathcal{MOVIE}(W)$ of distinct frames appearing on W.*

Figure 32 gives an example free assembly sequence α with a window W. The arrows between the edges on either side of W represent the relation "successor".

LEMMA 8.5. (WINDOW MOVIE LEMMA) *Let $\alpha \in \mathcal{H}^{\text{Free}}[\mathcal{T}, \sigma_A]$ and $\beta \in \mathcal{H}^{\text{Free}}[\mathcal{T}, \sigma_B]$ two assembly sequences. Assume there are two windows A in α and B in β and a translation τ such that $\tau(A) = B$, $\mathcal{MOVIE}(A)$ is the same as $\mathcal{MOVIE}(B)$ up to translation by τ, and lastly τ can be extended to a translation from* $\text{Far}(A)$ *to* $\text{Far}(B)$ *which maps $\sigma_B \cap \text{Far}(B)$ to $\sigma_A \cap \text{Far}(A)$.*

Let $\alpha^\dagger = \alpha_{\restriction \text{Far}(A)} \in \mathcal{H}^{\text{Free}}[\mathcal{T}, \lim \alpha \cap \text{Near}(A)]$ be the sequence of attachments of α within $\text{Far}(A)$*, and likewise $\beta^\dagger = \alpha_{\restriction \text{Far}(B)} \in \mathcal{H}^{\text{Free}}[\mathcal{T}, \lim \beta \cap \text{Near}(B)]$ be the sequence of attachments of β witihn* $\text{Far}(B)$.

Let α' be the candidate assembly sequence consisting of α where for every k, the attachment $\alpha'^\dagger_k$ is replaced with $\beta^\dagger_k$.

Then there is a window A' on α' and two translations $\tau_N : \text{Near}(A') \to \text{Near}(A)$ and $\tau_F : \text{Far}(A') \to \text{Far}(B)$ such that:

$$\begin{cases} \forall z \in \text{Near}(A'), & (\lim \alpha') \cdot \text{tile}(z) = (\lim \alpha) \cdot \text{tile}(\tau_N(z)) \\ \forall z \in \text{Far}(A'), & (\lim \alpha') \cdot \text{tile}(z) = (\lim \beta) \cdot \text{tile}(\tau_F(z)) \end{cases} \tag{8.1}$$

Moreover, if $\beta^\dagger >_{\text{fz}} \alpha^\dagger$, then $\alpha' >_{\text{fz}} \alpha$.

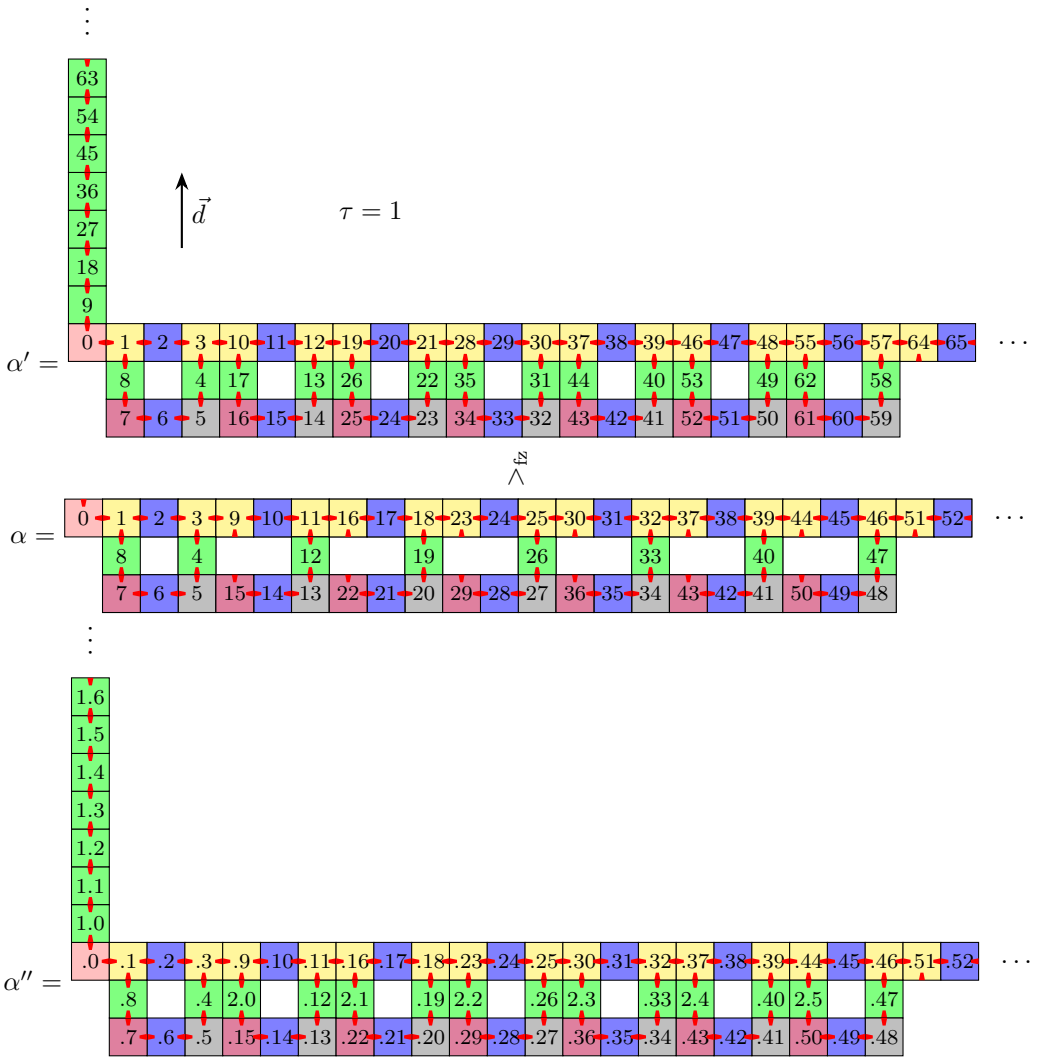


Figure 31: An assembly sequence α' of some temperature 1 aTAM, which yields a terminal production $F = \lim \alpha'$. Below a sequence $\alpha >_{\text{fz}} \alpha'$ which yields a smaller production P which is bounded in direction $\vec{d}$ (vertically). The assembly sequence α is fizzier since by step 50, it just closed its seventh hole while α' is lagging at only 5 holes. Because α skips any unprofitable attachment, $\lim \alpha$ is very much not terminal: its seed (tile number 0) as well as every tile attached at a time of the form $16 + 7k$ has an attachable yet unfilled position. It can thus be extended into an ordinal assembly sequence α'' which does reach $\lim \alpha$, but in time 3ω. The attachment times of the form $i.j$ in α'' should be read as $i\omega + j$. At time ω, α'' has assembled $\lim \alpha$; at time 2ω, it has added the upwards path, and at time 3ω, it has added the last tile to each hole of $\lim \alpha$.

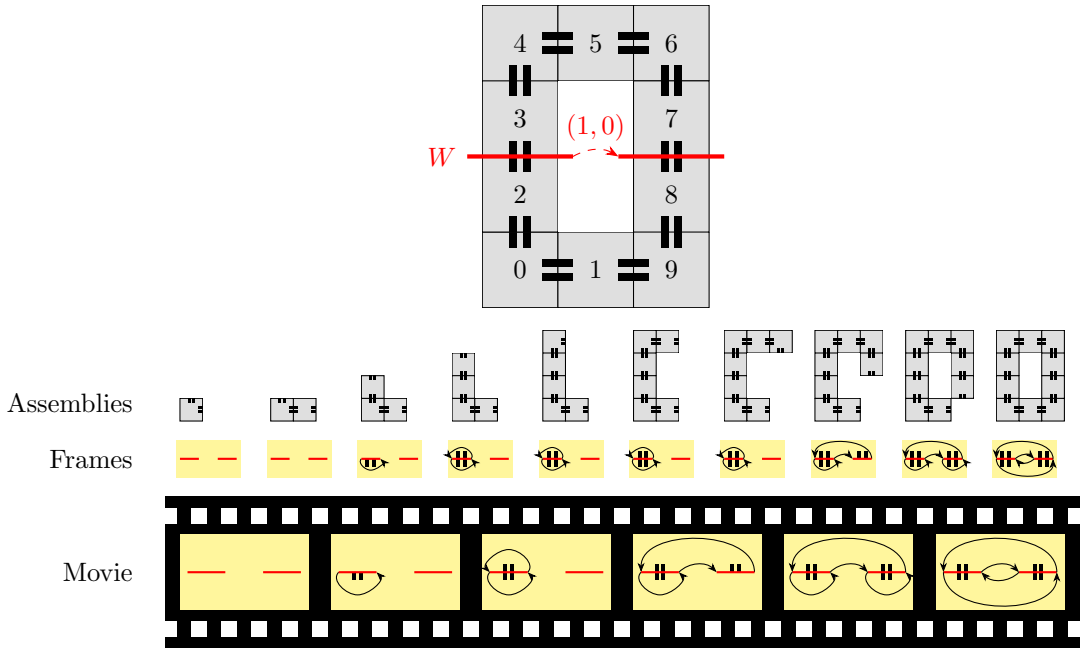


Figure 32: Example assembly sequence α and window W. On the main picture, the labels of the tiles of $\lim \alpha$ are the order in which they attach. The small pictures are the successive assemblies of α, their associated frames and the obtained movie. The movie is made up of the sequence of the unique frames, in order of apparition.

Proof. Define α' as α where for each k, the k-th attachment $t@Z$ within $\mathrm{Far}(A)$ has been replaced by the k-th attachment within $\mathrm{Far}(B)$ of β. When doing this, for an attachment $t@Z$ within $\mathrm{Far}(B)$, any arc $a = (v, d)$ in Z with $v \in \mathrm{Near}(B)$ is replaced by the arc $\tau^{-1}(a)$.

The translations τ_N and τ_F can be defined inductively on the attachments of α', such that Equation (8.1) holds.

The proof that α' is a valid free assembly sequence is the same as in the $\mathbb{Z}^2$ case. Note that since B is a cut of $(\lim \beta)\cdot$ support, any attachment of β in $\mathrm{Far}(B)$ can be replayed in α' in $\mathrm{Far}(A')$ without conflicts since they do not create any adjacency outside $\mathrm{Far}(\beta)$.

Observe that every attachment in $\mathrm{Far}(B)$ which affects holes in β affects the same number of holes in α': holes which are wholly within $\mathrm{Far}(B)$ have had all of their tiles attached in α', and holes which go through B become holes through A because the part within $\mathrm{Near}(B)$ has the same connections in $\mathrm{Near}(A)$ at that frame in the movie. ∎

The fact that the grafting process of Lemma 8.5 is increasing in the fizziness of the far part of the assembly sequences implies that when the original assembly sequences have maximal fizziness, so does the chimera sequence α'.

Lemma 8.5 will be most useful in this paper in the case where the cuts A and B are on the same branch of the assembly. In this case, by iterating Lemma 8.5, it is possible to get a production with a periodic subassembly.

Corollary 8.1. *Let $\alpha = (A_i)_{i<o}$ be an assembly sequence with two windows A and B satisfying the hypothesis of Lemma 8.5 and such that $\mathrm{Far}(B) \subset \mathrm{Far}(A)$. Let τ be the translation between A and B. Then, there is an assembly sequence α^τ such that $F = \lim \alpha^\tau \cap \mathrm{Far}(A)$ verifies $\tau(F) \subset F$, and within $\mathrm{Near}(B)$, α^τ does the same attachments in the same order as α.*

Proof. Let τ be the translation sending A to B.

Show by induction that for any k, there is a sequence α^k such that the movie on the windows A and B within α^k are the same as in α, and for each $z \in \mathrm{Near}(B) \cap \mathrm{Far}(A)$ and $j \leq k$, $(\lim \alpha^k)(\tau^j(z)) = (\lim \alpha)(z)$, and the attachments done by α^k and α^j within $\bigcup_{i\leq j} \tau^i(\mathrm{Near}(B) \cap \mathrm{Far}(A))$ are the same.

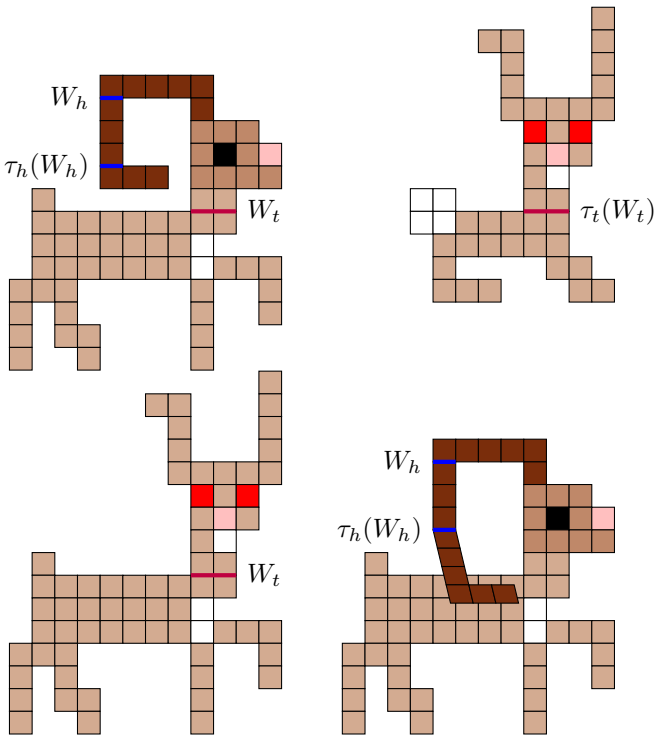


Figure 33: The Window Movie Lemma: consider two assembly sequences ι (the ibex) and β (the bunny), with their productions depicted in the top row. The sequence ι, has two windows W_t and W_h. In W_t, $\mathrm{Far}(W_t)$ is the head of the ibex, $\mathrm{Near}(W_t)$ its body. In W_h, $\mathrm{Far}(W_h)$ is the tip of the horn. Each of them is associated with a translation, τ_t and τ_h respectively, which sends the window to another window with the same movie, and satisfying the hypothesis of Lemma 8.5. Applying Lemma 8.5 in each case gives two new assembly sequences represented on the bottom row. Applying it with W_t yields a fearsome chimera, while its application to W_h yields an ibex with a horn so long it needs to bend in the third dimension to avoid piercing its spine, i.e. avoiding the conflict that would arise were our attachment sequences not free.

For $k = 0$, it suffices to pick $\alpha^0 = \alpha$. Assume that α^k satisfies the induction hypothesis. Then Lemma 8.5 applies, and the assembly sequence it yields, α^{k+1}, satisfies the induction hypothesis at rank $k+1$.

For a pair $(j,k) \in \mathbb{N}$, if for all $i \leq j$, $\alpha_i^k \in \bigcup_{l \leq k} \tau^k(\text{Near}(B) \cap \text{Far}(A))$, then the j first attachments are unchanging after rank k: for all $m \geq k$ and $i \leq j$, $\alpha_i^m = \alpha_i^k$. Let $\alpha^{\vec{p}}$ be the sequence of such unchanging attachments. Let m be the supremum of the $j \in \mathbb{N}$ such that the j first attachments are unchanging after some rank k. Define $\alpha^{\vec{p}}$ as the sequence of length m with $\alpha_j^{\vec{p}} = \alpha_j^k$, where k is such that the j first attachments are unchanging.

It is easy to check that indeed, $\lim \alpha^{\vec{p}} \cap \text{Far}(A)$ has period $\vec{p}$: any attachment in $\alpha^{\vec{p}}$ within $\text{Far}(A)$ gets picked up by subsequent applications of Lemma 8.5 which translate it by $\vec{p}$. ∎

In this situation with a branch and two cuts with the same movie, it is also possible to cut assembly sequences short, so that they can do their business in $\text{Near}(A)$ without needing to mess with $\text{Far}(B)$.

COROLLARY 8.2. *Let α be an assembly sequence with two windows A and B satisfying the hypothesis of Lemma 8.5 and such that* $\text{Far}(B) \subset \text{Far}(A)$. *Then there is an assembly sequence α' which does the same attachments as α within* $\text{Near}(A)$, *but has no attachment in* $\text{Far}(B)$.

Proof. Let j be the last time that α does an attachment in $\text{Near}(A)$ next to A, i.e. the date of the last frame in the movie of A where a tile is attached on the near side of the window. By Remark 1, up to a reordering of α, j is finite.

Let α^0 be the prefix of length j of α. For each k, if α^k has any attachment in $\text{Far}(B)$, it is possible to use Lemma 8.5 between B and A to obtain a sequence α^{k+1} with fewer attachments within $\text{Far}(B)$. Hence, there is a finite k such that α^k has no attachments in $\text{Far}(B)$, and $\lim \alpha^k \cap \text{Near}(A) = \lim \alpha^0 \cap \text{Near}(A)$. Since by time j, the movie on A is over, the rest of the attachments of α which take place in $\text{Near}(A)$ can be replayed after α^k. ∎

8.5 The Tree Pump Theorem After all these considerations about the fantastic properties of Ordinal Free Assembly Sequences, it is now time to come back to Earth, or rather $\mathbb{Z}^2$. The object is now to prove theorem 3.4, for which an investigation of the properties of systems whose $\mathbb{Z}^2$-assembly sequences do not circle large squares is in order.

The statement of theorem 3.4 is given again, recall that it deals with the $\mathbb{Z}^2$-productions of the aTAM system $\mathcal{T}$, hence in its statement, $\mathcal{A}_\square[\mathcal{T}]$, $C_m[\mathcal{T}]$, $B_{F(n,m),\vec{d}}[\mathcal{T}]$ and $P_{\vec{d}}[\mathcal{T}]$ are sets of $\mathbb{Z}^2$-assemblies.

THEOREM 8.2. (TREE PUMP) *For any aTAM system $\mathcal{T} = (S, \sigma, \tau)$ with σ finite and connected, define the following sets of assemblies:*

- *for any integer m, $C_m[\mathcal{T}]$ is the set of assemblies of $\mathcal{T}$ which encircle an $m \times m$ square;*
- *for any real k and vector $\vec{d}$, $B_{k,\vec{d}}[\mathcal{T}]$ is the set of assemblies of $\mathcal{T}$ which do not cover any position $\vec{p}$ such that $\vec{p} \cdot \vec{d} > k + |\text{dom } \sigma|$*
- *for any vector $\vec{d}$, $P_{\vec{d}}[\mathcal{T}]$ is the set of* ultimately periodic *assemblies A of $\mathcal{T}$ such that there is a vector $\vec{p}$ with $\vec{p} \cdot \vec{d} > 0$ and a non-empty sub-assembly $a \sqsubseteq A$ such that $a + \vec{p} \sqsubseteq a$.*

Then, there is a function $F : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that for aTAM system $\mathcal{T}$ with n tile types and a 1-tile seed, integer m and unit vector $\vec{d}$ of $\mathbb{R}^2$,

$$\mathcal{A}_\square[\mathcal{T}] \cap (C_m[\mathcal{T}] \cup B_{F(n,m),\vec{d}}[\mathcal{T}] \cup P_{\vec{d}}[\mathcal{T}]) \neq \varnothing.$$

When considering the statement of theorem 3.4, the holes of the $\mathbb{Z}^2$-productions of $\mathcal{T}$ might as well be filled in. Hence, the definition of their fill-in, illustrated on Figure 34.

DEFINITION 8.12. (FILL-IN) *For a subgraph $D \subset \mathbb{Z}^2$, define the* fill-in $D^\bullet$ *as the subgraph of $\mathbb{Z}^2$ induced by the positions p such that there is no infinite path from p in $\mathbb{Z}^2 \setminus D$.*

A $\mathbb{Z}^2$ assembly A which does not circle any square larger than $m \times m$ looks like a tree, as expressed by the notion of *Connected Treewidth* [9]. This tree is obtained by grouping the positions of A· support in connected sets of size at most $2m$, known as *bags*, organized in a tree in such a way that:

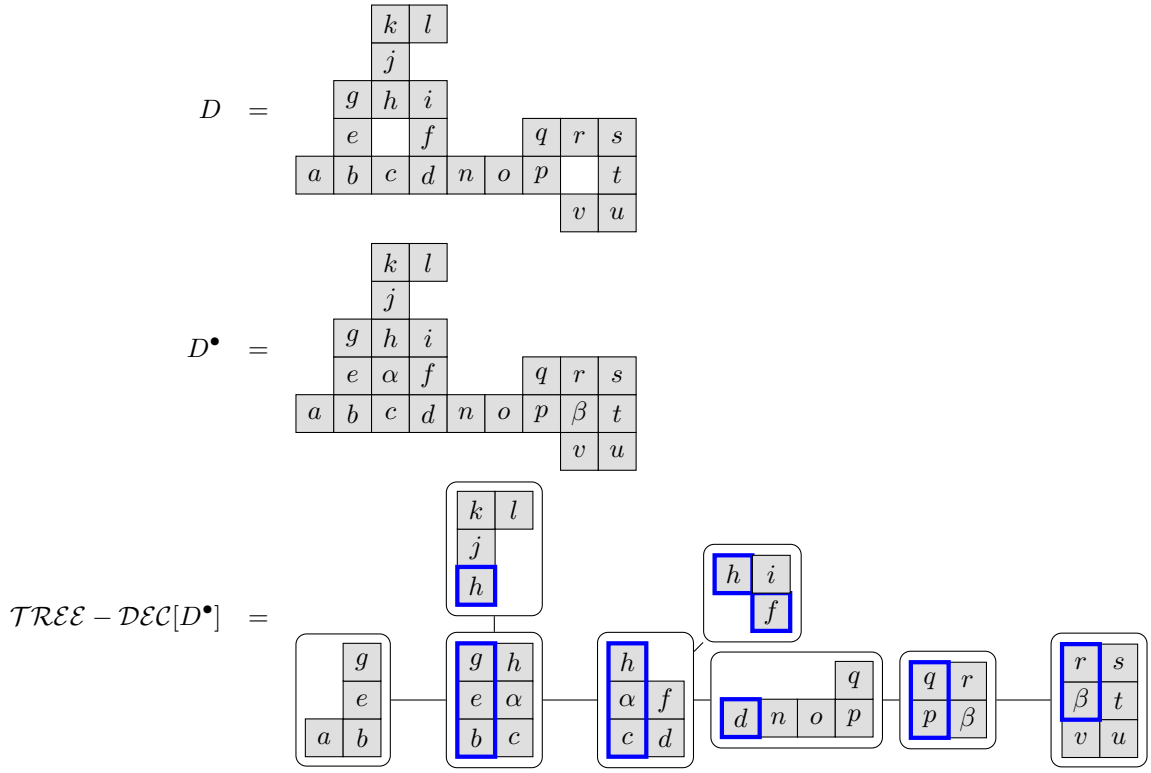


Figure 34: The domain D of a production P, its fill-in $D^\bullet$ and an associated connected tree decomposition of $\mathcal{TREE} - \mathcal{DEC}[D^\bullet]$. The blue part of each bag b of $\mathcal{TREE} - \mathcal{DEC}[D^\bullet]$ is its intersection W_b with its parent, which disconnects the vertices appearing in its subtree from the rest of $D^\bullet$. The choice of the root of $\mathcal{TREE} - \mathcal{DEC}[D^\bullet]$ is arbitrary and does not affect the sets W_b, up to renaming.

- any arc of $A\cdot$support is between two positions which appear in the same bag, and
- for any position $z \in A\cdot$support, the set of bags which contain z forms a subtree.

This decomposition is represented on Figure 34.

LEMMA 8.6. *Let $\mathcal{T}$ an aTAM system, m an integer and A a $\mathbb{Z}^2$ assembly of $\mathcal{T}$ such that $A \notin C_m[\mathcal{T}]$. Then $A\cdot$*support *has Connected Tree Width $2m$.*

Proof. Indeed, it has treewidth m: otherwise it would contain an $m \times m$ grid as a minor; that minor would have to be realized in $\mathbb{Z}^2$ as a subgraph which would encircle some $m \times m$ square. Moreover, since $P^\bullet$ does not encircle any empty position, it does not have any *geodesic cycle* of length more than 4. ∎

The point of using Connected Treewidth rather than the usual treewidth is that thanks to the connectivity of the bags of $\mathcal{TREE} - \mathcal{DEC}[P^\bullet]$, the distances in $\mathcal{TREE} - \mathcal{DEC}[P^\bullet]$ reflect those in $P^\bullet$:

- there is a function r such that for any vertex $v \in P^\bullet$, the subtree of bags of $\mathcal{TREE} - \mathcal{DEC}[P^\bullet]$ containing v has a size at most $r(m)$,
- there are two increasing functions d_m, D_m such that for any vertices u, v at distance δ in $P^\bullet$, any bags $B_u \ni u$ and $B_v \ni v$ of $\mathcal{TREE} - \mathcal{DEC}[P^\bullet]$ are at distance Δ with $d_m(\delta) \leq \Delta \leq D_m(\delta)$.

As a consequence, if two tiles are far enough in an assembly A and the path between them does not go through the seed, there must be two windows cutting that path which satisfy the hypothesis of Corollary 8.1.

LEMMA 8.7. *Let $\mathcal{T}$ be an aTAM system with n tiles and m an integer.*

Let $\alpha = (A_i)_{i\leq o} \in \mathcal{H}^{\text{Free}}[\mathcal{T}, A_0]$ be such that for all $i \leq o$, $\text{squash}(A_i) \notin C_m[\mathcal{T}]$.

There is a constant $F(n,m)$ such that if $z, z' \in \text{squash}(\lim\alpha)\cdot\text{support}\setminus A_0\cdot\text{support}$ are such that the distance between z' and z is greater than $F(n,m)$, then there are two windows A and B with $z \in \text{Near}(A)$ and $z' \in \text{Far}(B)$ which satisfy the hypothesis of Lemma 8.5.

Proof. let $W(n,m)$ be the number of movies with n glues on a window of width $2m$. Note that $W(n,m)$ counts the number of arrangements of the edges within the window, the perspective difference between the connected components of the window, as well as the events on the frames of the movie. Let $F(n,m) = d_m^{-1}(W(n,m))$.

Let $P = \lim\alpha\cdot\text{support}$. By Lemma 8.6, pick a tree decomposition $\mathcal{TRECE} - \mathcal{DEC}[P^\bullet]$ with connected bags of size at most $2m$. Let δ be the distance between z and z'. If $\delta \leq F(n,m)$, by the pigeonhole principle, there must be two windows A and B between z and z' with the same movie. ∎

In the case where the assembly embeds into $\mathbb{Z}^2$, one actually controls the direction of the vector between the two windows.

LEMMA 8.8. *Let $\mathcal{T}$ be an aTAM system with n tiles, m an integer and $\vec{d}$ a unit vector.*

Let $\alpha = (A_i)_{i\leq o} \in \mathcal{H}^{\text{Free}}[\mathcal{T}, A_0]$ be such that for all $i \leq o$, A_i embeds into $\mathbb{Z}^2$.

There is a constant $F'(n,m)$ such that if $z, z' \in \lim\alpha\cdot\text{support}\setminus A_0\cdot\text{support}$ are such that the distance between $(z'-z)\cdot\vec{d} > F(n,m)$, then there are two windows A and B with $z \in \text{Near}(A)$ and $z' \in \text{Far}(B)$ which satisfy the hypothesis of Lemma 8.5, and such that the vector $\vec{p}$ of the translation between A and B satisfies $\vec{p}\cdot\vec{d} > 1$.

Proof. Pick $F'(n,m) = 2F(n,m)+1$. If $(z'-z)\cdot\vec{d} > F'(n,m)$, then in $\mathcal{TRECE} - \mathcal{DEC}[\lim\alpha\cdot\text{support}^\bullet]$ there there are two bags, one b containing z, the other $b' \ni z'$ such that on the path between b_z and $b_{z'}$, there are $2F(n,m)+1$ bags $(b_i)_i$ such that for all $x \in b_i, y \in b_j, (x-y)\cdot\vec{d} \geq i-j$. Thus, there are i,j such $j > i+1$ and two windows, A between b_i and b_{i+1}, and B between b_j and b_{j+1} which have the same movie. Because the windows A and B are separated by at least two elements of (b_i), the translation vector $\vec{p}$ between them satisfies $\vec{p}\cdot\vec{d} > 1$. ∎

Thus, there is a simple argument to prove theorem 3.4: pick a sequence of maximal fizziness which produces a terminal assembly of $\mathcal{T}$ which is neither in $B_{F'(n,m),\vec{d}}[\mathcal{T}]$ nor in $C_m[\mathcal{T}]$. Lemma 8.8 yields two windows with the same movie; by applying Corollary 8.1 between them, a production in $P_{\vec{d}}[\mathcal{T}]$ appears. Alas, the sequence on which this argument is founded may simply fail to exist: the ω-sequences of maximal fizziness may fail to reach a terminal production. One could extend them in order to reach a terminal production, but there may not be a sequence of maximal fizziness among these (ordinal) extensions.

Thus, it is necessary to prevent the pesky tendency ordinal sequences have to try and buy time by spacing out their attachment so as to generate an infinite family of sequences with increasing fizziness. *Straight* sequences are the answer: they do not have the wiggle room to misbehave.

DEFINITION 8.13. *Let $\mathcal{T}$ be an aTAM system, and $\prec$ an arbitrary total order on its sequences. A free, ordinal assembly sequence $\alpha = (A_i)_{i<o} \in \mathcal{H}^{\text{Free}}[\mathcal{T}]$ is $\prec$-*straight *if for every pair of window N, F such that N and F have the same movie and $A_0\cdot\text{support} \subset \text{Near}(N)$, α is the smallest sequence for $\prec$ obtained by applying Corollary 8.1 to N and F in α.*

The order $\prec$ is henceforth fixed and left implicit.

LEMMA 8.9. *Let $\mathcal{T} = (S, \sigma, \tau)$ be an aTAM system and w an integer.*

Let $X \subset \mathcal{H}^{\text{Free}}[\mathcal{T}]$ be a set of assembly sequences, and d an integer. If for any sequence $\alpha \in X$ and any position $z \in \lim\alpha\cdot\text{support}$ at distance more than d from σ, there are two windows A and B such that $\sigma\cdot\text{support} \subset \text{Near}(A)$ and $z \in \text{Far}(B)$, then there is a finite number of straight sequences in X.

Proof. In this case, each straight sequence in X is determined by what it does in a radius d around σ. ∎

LEMMA 8.10. *let $\mathcal{T} = (S, \sigma, \tau)$ be an aTAM system and α a straight assembly sequence of $\mathcal{T}$. There is a straight assembly sequence α' which extends α such that $\lim\alpha' \in \mathcal{A}_\square[\mathcal{T}]$.*

Proof. Let $P = \lim \alpha$. If $P \notin \mathcal{A}_\square[\mathcal{T}]$, then there is an attachment $t@Z$ which is possible in P.

Let α' be α with $t@Z$ appended at its end, and z the position of that last attachment in $\lim \alpha' \cdot$ support. Assume that Z is such that the distance between $\sigma \cdot$ support and z is minimal.

If there are no pairs of windows (N, F) in α' such that $\mathrm{Far}(F) \subset \mathrm{Far}(N)$, $\mathcal{MOVIE}(N) = \mathcal{MOVIE}(F)$, $\sigma \cdot$ support $\subset \mathrm{Near}(N)$ and $z \in \mathrm{Far}(N)$, then α' is straight.

If there are two windows (N, F) in α' such that $\mathrm{Far}(F) \subset \mathrm{Far}(N)$, $\mathcal{MOVIE}(N) = \mathcal{MOVIE}(F)$, $\sigma \cdot$ support $\subset \mathrm{Near}(N)$ and $z \in \mathrm{Far}(F)$, then by Corollary 8.2, there is a place closer to σ than z where an attachment was possible.

Lastly, if there is a pair of windows (N, F) in α' such that $\mathrm{Far}(F) \subset \mathrm{Far}(N)$, $\mathcal{MOVIE}(N) = \mathcal{MOVIE}(F)$, $\sigma \cdot$ support $\subset \mathrm{Near}(N)$ and $z \in \mathrm{Far}(N) \cap \mathrm{Near}(F)$, then the sequence α'' obtained by applying Corollary 8.1 in α' is straight and has α as a prefix, since the new attachments (the repetitions of $t@Z$) are done last in each repetition of the movie in $\mathrm{Far}(N) \cap \mathrm{Near}(F)$.

This process yields a straight assembly sequence α' which strictly extends α. It can be repeated until $\lim \alpha' \in \mathcal{A}_\square[\mathcal{T}]$. ∎

LEMMA 8.11. *Let $\mathcal{T} = (S, \sigma, \tau)$ be an aTAM system and α a straight assembly sequence of $\mathcal{T}$. Then* $\mathrm{squash}(\alpha)$ *is a prefix of a straight sequence α'. Moreover, the connected treewidth of* $\lim \alpha' \cdot$ support *is no greater than that of* $(\lim \mathrm{squash}(\alpha)) \cdot$ support.

Proof. If α embeds cleanly in $\mathbb{Z}^2$, there is nothing to prove since $\mathrm{squash}(\alpha) = \alpha$.

Otherwise, $\mathrm{squash}(\alpha)$ stops because one of its attachment $t@z$ creates a hole which does not exist in α. Because α is straight, there cannot be a pair of windows with the same movie separating σ and z. Applying Corollary 8.1 in $\mathrm{squash}(\alpha)$ on each pair of windows with the same movie which are the closest to σ yields a straight sequence α' extending $\mathrm{squash}(\alpha)$.

Fix a tree decomposition of $(\lim \mathrm{squash}(\alpha)) \cdot$ support. Because α is straight, among all tree decompositions of $\lim \alpha' \cdot$ support, there is one where each bag which lies in the far side of a pair of windows is a translation of some bag close to the seed. This tree decomposition has the same width as the decomposition of $(\lim \mathrm{squash}(\alpha)) \cdot$ support. ∎

At last, all the ingredients are there to prove theorem 3.4.

Proof. [Proof of theorem 3.4] Define a sequence (α_i) of straight assembly sequences as follows:

- Fix α_0 to be the assembly sequence with zero attachments: $\alpha_0 = (\sigma)$;
- for i even, α_{i+1} is a straight extension of α_i which reaches a terminal production;
- for i odd, α_{i+1} is obtained from α_i by Lemma 8.11.

For every i, either $\alpha_{i+1} = \alpha_i$ or $\alpha_{i+1} >_{\mathrm{fz}} \alpha_i$: at the even steps, if $\alpha_{i+1} \neq \alpha_i$, then it is a proper extension and is thus more fizzy; at the odd steps, if α_i does not embed into $\mathbb{Z}^2$, then $\mathrm{squash}(\alpha_i)$ is more fizzy, and so is its extension α_{i+1}.

If $\mathcal{A}_\square[\mathcal{T}] \cap C_m[\mathcal{T}] = \varnothing$, then for each i odd, the Connected Tree-width of $\mathrm{squash}(\alpha_i)$ is at most $2m$, hence the Connected Tree-width of α_{i+1} is at most $2m$. But then, by Lemma 8.9, α_i can only take a finite number of different values for i even.

Thus, the sequence (α_i) is eventually stationary, and its fixed point β is a straight sequence which embeds into $\mathbb{Z}^2$ and reaches a terminal production. If $\mathcal{A}_\square[\mathcal{T}] \cap B_{F(n,m),\vec{d}} \neq \varnothing$, this terminal production must reach at least $F(n, m)$ in direction $\vec{d}$. Thus $\lim \beta \cdot$ support has a branch with two windows A, B such that $\mathcal{MOVIE}(A) = \mathcal{MOVIE}(B)$ and the vector sending A to B verifies $\vec{p} \cdot \vec{d} > 1$ (by Lemma 8.8). Because β is straight, that branch is periodic, thus $\lim \beta \in \mathcal{A}_\square[\mathcal{T}] \cap P_{\vec{d}}$. ∎

9 From an aTAM Quine to a Self-Assembled Discrete Self-Similar Fractal

In this section, we present our first fractal construction, which is an aTAM system that performs an infinite series of nested simulations of itself, at greater and greater scale factors. Importantly, unlike previous IU constructions (e.g. [10, 23, 22]), this construction begins with a seed consisting of a single tile. That tile first grows into an

$m \times m$ macrotile whose sides have a full definition of the system and the glues of the seed tile on its output sides, using the construction of the proof for Theorem 3.2. Then, since that portion of the system has been designed so that it meets the requirements of a standard TAS, the tile set that is IU for the class of standard aTAM systems treats that macrotile as a scaled version of the seed tile, and simulates the growth of the original $m \times m$ macrotile but with each tile represented by an $m \times m$ macrotiles, resulting in an $m^2 \times m^2$ macrotile. By combining the tilesets of the two results, we are able to cause this process of simulation at increasing scale factor to happen for an infinite series of scale factors.

Then, to cause the resulting assembly to be a discrete self-similar fractal with ζ-dimension < 2, we make slight modifications to the previous IU and quine constructions so that the macrotiles contain a specified amount of empty space. Then, the repeated simulation at greater and greater scales yields a DSSF.

We prove Theorem 3.5 by construction. Given an arbitrary $z, \epsilon \in \mathbb{Q}$, where $1 < z < 2$ and $\epsilon < 1$, we demonstrate an aTAM system $\mathcal{F}_{z,\epsilon} = (T, \sigma, 2)$ that strictly self-assembles a DSSF whose ζ-dimension is $z \pm \epsilon$. Our construction utilizes the constructions from the proofs of Theorems 3.1 and 3.2 with slight modifications. Those modifications and the proofs of their correctness, and thus the overall correctness of the proof of Theorem 3.5, are included in subsections dedicated to each of the following:

1. Modification to allow simulation to occur at greater and greater scales instead of stopping after first level.
2. Proof that simulation is correct at each level.
3. Modifications to the quine macrotile and the IU system's macrotiles to add spacing for targeted fractal dimensions.
4. Derivation of the resulting fractal dimension.

9.1 Nested self-simulation at an infinitely increasing series of scales The aTAM quine $\mathcal{Q} = (Q, \sigma, 2)$ presented in the proof of Theorem 3.2 creates a finite terminal assembly, which we'll call α_Q. As it is a quine with respect to the tileset U (the tileset that is IU for the class of standard aTAM systems) and its representation and seed generation functions R and S, α_Q is a macrotile mapping to σ under R (and having the same shape and exterior glues as $S(\sigma)$). This in turn means that, if α_Q is used as the seed for a system containing the tiles of U (and $\tau = 2$ since both $\mathcal{Q}$ and U utilize $\tau = 2$), which we'll call $\mathcal{Q}^+$ (i.e. $\mathcal{Q}^+ = (U, \alpha_Q, 2)$), then $\mathcal{Q}^+$ would simulate $\mathcal{Q}$ under R (at some scale factor m) and its terminal assembly, α_{Q^+}, would be α_Q scaled by m (as interpreted under R). Additionally, there would be two macrotile locations that map to empty space under R but which contain fuzz (which does not compromise the correctness of the simulation). Let c be the dimensions of α_Q, which is a square. The macrotile locations in $\mathcal{Q}^+$ containing the fuzz would be those mapping to the locations along the counter-clockwise-most encoding of the strength-2 glue being presented by the simulated macrotile. This is purely a chosen convention though and there is no particular significance to this location over any other. Thus, in the scaled simulation of $\mathcal{Q}^+$, macrotile growth would be initiated in those locations (because strength-2 glues are simulated there), but the growth by the tiles of U that attempts to find a match in the glue lookup table would fail because the tiles of U are not encoded in it. This would result in terminal growth that is valid fuzz, mapping only to empty space.

In order to create $\mathcal{F}_{z,\epsilon}$ so that the simulation continues indefinitely at all scales, we simply modify the procedure used to create $\mathcal{Q}$ by including the tile types from U in T_F in the glue table. With the tile types of U also encoded in the glue lookup table, the simulation will continue where $\mathcal{Q}^+$ terminated. The following section details that the relation of intrinsic simulation is transitive and, consequently, this will result in the increasingly higher order simulations of Q at an infinite series of larger scales and thus a DSSF.

9.2 Correctness of nested self-simulation at infinitely increasing scales Here we prove that having equivalent productions is transitive.

LEMMA 9.1. *Let $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$ be TAS's satisfying $\mathcal{S} \Leftrightarrow_{R_1} \mathcal{T}$ and $\mathcal{T} \Leftrightarrow_{R_2} \mathcal{U}$, then $\mathcal{S} \Leftrightarrow_{R_2 \circ R_1} \mathcal{U}$.*

Proof. By hypothesis $\mathcal{A}[\mathcal{T}] = R_1^*(\mathcal{A}[\mathcal{S}])$ and $\mathcal{A}[\mathcal{U}] = R_2^*(\mathcal{A}[\mathcal{T}])$. Thus $R_2^*(R_1^*(\mathcal{A}[\mathcal{S}])) = R_2^*(\mathcal{A}[\mathcal{T}]) = \mathcal{A}[\mathcal{U}]$. The same holds true for the sets of terminal assemblies.

It must also be the case that assemblies in $\mathcal{S}$ map cleanly to those in $\mathcal{U}$ since all fuzz locations in $\mathcal{T}$-assemblies R_2-mapping to $\mathcal{U}$-assemblies are fuzz locations in the $\mathcal{S}$-assemblies R_1-mapping to the $\mathcal{T}$-assemblies. ∎

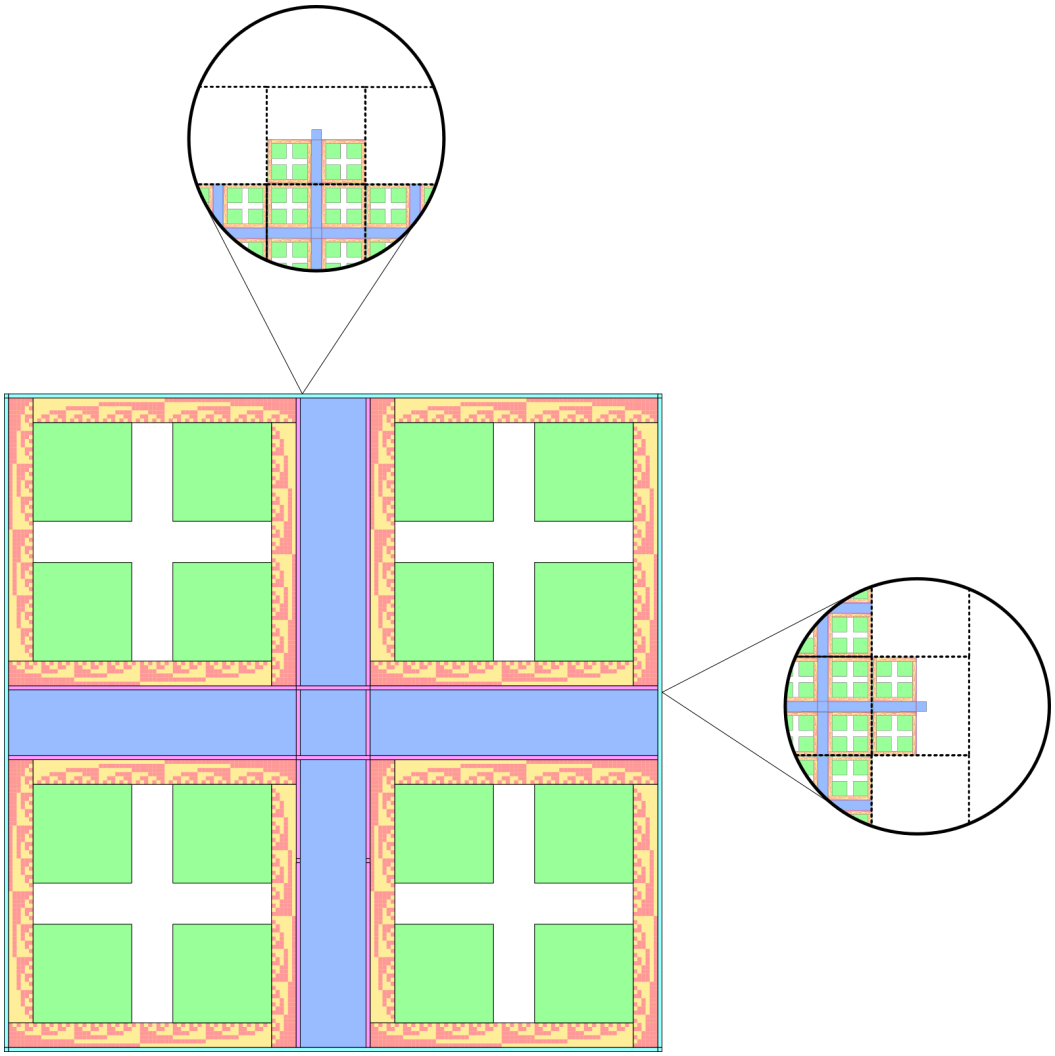


Figure 35: Without encoding the IU tileset in the glue table, Q^+ will grow up to the point where the quine has acted as the seed for a scaled copy of itself. Note that the large macrotile illustrated here is an order 2 structure made from macrotiles itself. In other words, Q^+ simulates Q using macrotiles. The zoomed-in views on the north and east side depict the fuzz macrotile locations that will contain partial macrotiles. These correspond to the positions along the side of Q where control is given to the IU tileset, however growth stops here because the glue tables on the tiles in Q^+ do not contain encodings for the individual tiles in the IU tileset U. These will not map to tiles under the representation function since the glue table failed to find a match, so Q^+ shows that Q acts as a seed for itself.

This result says that the follows relationship is transitive

LEMMA 9.2. *If $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$ are TAS's satisfying $\mathcal{T} \dashv_{R_1} \mathcal{S}$ and $\mathcal{U} \dashv_{R_2} \mathcal{T}$, then $\mathcal{U} \dashv_{R_2 \circ R_1} \mathcal{S}$.*

Proof. Let $\alpha'', \beta'' \in \mathcal{A}[\mathcal{S}]$ such that $\alpha'' \to^{\mathcal{S}} \beta''$. We know that $R_1^*(\alpha'') \to^{\mathcal{T}} R_1^*(\beta'')$ by hypothesis. Furthermore, let $\alpha' = R_1^*(\alpha'')$ and let $\beta' = R_1^*(\beta'')$. By hypothesis, since $\alpha' \to^{\mathcal{T}} \beta'$, it must be the case that $R_2^*(\alpha') \to^{\mathcal{U}} R_2^*(\beta')$. Consequently, $R_2^*(R_1^*(\alpha'')) \to^{\mathcal{U}} R_2^*(R_1^*(\beta''))$ and the lemma is proved. ∎

LEMMA 9.3. *If $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$ are TAS's satisfying $\mathcal{S} \vDash_{R_1} \mathcal{T}$ and $\mathcal{T} \vDash_{R_2} \mathcal{U}$, then $\mathcal{S} \vDash_{R_2 \circ R_1} \mathcal{U}$.*

Proof. Let $\alpha \in \mathcal{A}[\mathcal{U}]$ and let $\Pi_\alpha^{\mathcal{T}} \subset \mathcal{A}[\mathcal{T}]$ be the stem set of α under R_2. For each element $\alpha' \in \Pi_\alpha^{\mathcal{T}}$, let $\Pi_{\alpha'}^{\mathcal{S}}$ be the corresponding stem set of α' under R_1 in $\mathcal{A}[\mathcal{S}]$. Now we define $\Pi_\alpha^{\mathcal{S}} = \bigcup_{\alpha' \in \Pi_\alpha^{\mathcal{T}}} \Pi_{\alpha'}^{\mathcal{S}}$ to be the union of these stem sets. We will now show that $\Pi_\alpha^{\mathcal{S}}$ is the stem set of α under $R_2 \circ R_1$.

By definition, if $\alpha'' \in \Pi_\alpha^{\mathcal{S}}$, then $\alpha'' \in \Pi_{\alpha'}^{\mathcal{S}}$ for some $\alpha' \in \Pi_\alpha^{\mathcal{T}}$. Consequently, $R_1^*(\alpha'') \in \Pi_\alpha^{\mathcal{T}}$ and thus $R_2^*(R_1^*(\alpha'')) = \alpha$. Now, let $\beta \in \mathcal{A}[\mathcal{U}]$ such that $\alpha \to^{\mathcal{U}} \beta$. We will show that the two conditions in the definition of *models* hold.

For the first condition, let $\alpha'' \in \Pi_\alpha^{\mathcal{S}}$. By definition, $\alpha'' \in \Pi_{\alpha'}^{\mathcal{S}}$ for some $\alpha' \in \Pi_\alpha^{\mathcal{T}}$. Furthermore, by definition, there exists some $\beta' \in R_2^{*-1}(\beta)$ such that $\alpha' \to^{\mathcal{T}} \beta'$ and there exists some $\beta'' \in R_1^{*-1}(\beta')$ such that $\alpha'' \to^{\mathcal{S}} \beta''$. Since $\beta'' \in R_1^{*-1}(\beta')$ and $\beta' \in R_2^{*-1}(\beta)$, we may conclude that $\beta'' \in R_1^{*-1}(R_2^{*-1}(\beta))$

For the second condition, let $\alpha'' \in R_1^{*-1}(R_2^{*-1}(\alpha))$ and $\beta'' \in R_1^{*-1}(R_2^{*-1}(\beta))$ with $\alpha'' \to^{\mathcal{S}} \beta''$. We want to say that $\Pi_\alpha^{\mathcal{S}}$ contains an assembly which $\mathcal{S}$-produces α''. Because $\mathcal{T}$ models $\mathcal{U}$ we know that there exists an assembly which $\mathcal{T}$-produces $R_1(\alpha'')$ in the corresponding stem set $\Pi_{\alpha'}^{\mathcal{S}}$. The conclusion then follows by condition 2 of $\mathcal{S}$ modelling $\mathcal{T}$. ∎

THEOREM 9.1. *If $\mathcal{S}$ intrinsically simulates $\mathcal{T}$ and $\mathcal{T}$ intrinsically simulates $\mathcal{U}$, then $\mathcal{S}$ intrinsically simulates $\mathcal{U}$.*

This result follows from the previous lemmas. It is also clear that the scale factor of the intrinsic simulation of $\mathcal{U}$ by $\mathcal{S}$ is the product of the component scale factors.

To see why $\mathcal{F}_{z,\epsilon}$ strictly assembles into a DSSF, we reiterate some properties of our construction. First, our Quine tileset will grow from a single tile seed σ into a square α_σ^1 which resembles a macrotile for our IU tileset. This macrotile will have all of the information needed for our IU tileset to simulate the growth from σ to α_σ^1 using macrotiles instead of individual tiles. The result will be a larger assembly α_σ^2 whose shape consists each occupied tile location in α_σ^1 substituted by the shape of a macrotile. Importantly, our IU tileset is configured to simulate tiles using macrotiles so that each has a shape identical to that of α_σ^1, and furthermore, without the need for fuzz so long as all tiles are included in the glue table encoding. Consequently, since all of the IU tiles are encoded into $\mathcal{F}_{z,\epsilon}$ so that the glue table along α_σ^1 contain the IU tileset entries, there is nothing stopping the tileset from growing from the individual seed σ into α_σ^2. Since $\mathcal{F}_{z,\epsilon}$ intrinsically simulates itself at some scale, transitivity implies that it simulates itself at all powers of that scale.

9.3 Adding space to macrotiles Since the quine assembly will act as the seed macrotile for the simulation of itself by U, the exact shape of the square, will act as a generator for a DSSF and influence the ζ-dimension of the resulting assembly. Here we present a scheme for "squaring" the result from our quine in such a way that the ζ-dimension of the eventual DSSF may be effectively freely chosen.

Let W and H be the width and height respectively of the rectangle produced by the Quine tile set. Note that both of these numbers depend on the number of tile types to be encoded in the representation produced by the Quine. Specifically, if the Quine produces the representation of a tile set T with t distinct tile types, then both W and H grow as $\Theta(t \log_2 t)$.

Each square frame is made from 4 binary counter gadgets which grow into a closed loop. The length of each of these counters may be chosen arbitrarily in advance, so let X be the chosen length. Therefore, each counter that makes up the frame will effectively grow into an X by $\lceil \log_2 X \rceil$ rectangle. The side length of each square frame is therefore $F = X + \lceil \log_2 X \rceil$ and the entire macrotile square will therefore end up being $2F + W$ tiles on each side.

Within each of the square frames, 4 solid square blocks of tiles grow from each interior corner to fill in some of the space within each frame. The side length of these squares is precisely Y tiles where Y is some counter value chosen in advance. The number of tiles within each of these blocks, its area, is therefore Y^2. Furthermore, the

tiled area of each square frame is therefore $4(X\lceil\log_2 X\rceil + Y^2)$. Since the area of the Quine is $W \cdot H$ and the area of the rectangles propagating the Quine information to the north and east is $W(X + \lceil\log_2 X\rceil)$, we can compute the tiled area of the entire square macrotile to be

$$A = 16(X\lceil\log_2 X\rceil + Y^2) + 2W(X + \lceil\log_2 X\rceil) + WH$$

It's important to note that there's a bit of an awkward dependency here; the width W depends on the number of tile types to be encoded by the Quine, and in order to change the value of X or Y, the number of tile types to be encoded must change. In other words, increasing X or Y will result in an increase to W as well. In order to handle this, we may divide our overall tileset into two parts, one which has only those tiles that encode the values of X and Y which we will call $T_{X,Y}$ and all of the other tiles including those for the Quine, IU simulation, and square frame which we will call T_{else}. Both X and Y are encoded as binary numbers so the size of $T_{X,Y}$ is simply $\lceil\log_2 X\rceil \cdot \lceil\log_2 Y\rceil$. We can make a simplification here by noting that Y will never be larger than half of X and so it's bit representation will be no longer than that of X. Therefore, we can safely say that the number of tiles in $T_{X,Y}$ is $\Theta(\log_2 X)$. Since $|T_{else}|$ is simply a constant, the size of W is therefore $\Theta(\log_2 X \cdot \log_2 \log_2 X)$. The same holds for H. Combining these results, we find that the side length of the resulting macrotile square is

$$S = 2(X + \lceil\log_2 X\rceil) + \Theta(\log_2 X \cdot \log_2 \log_2 X)$$

while the area occupied by tiles in the square macrotile will be

$$A = 16(X\lceil\log_2 X\rceil + Y^2) + \Theta(X \cdot \log_2 X \cdot \log_2 \log_2 X)$$

9.4 Derivation of the resulting fractal dimension As shown in [19], for a DSSF with a generator G which has a minimum bounding box that is square with side length s, we know that the ζ-dimension of the DSSF will be

$$\frac{\log_2 |G|}{\log_2 s}$$

In our construction the Quine macrotile square acts as the generator for our DSSF, so the side length of the bounding box s is just S while the number of tile locations occupied in the generator $|G| = A$. Consequently, the ζ-dimension of our resulting DSSF will be

$$\frac{\log_2(A)}{\log_2(S)}$$

Attempting to plug in the previously deduced values of A and S, we could find a messy expression for the exact ζ-dimension given our values of X and Y and the total number of tile types in the tile set to be simulated. Instead, we note that something interesting happens when the limit is taken as X grows to infinity. We show that we can make the ζ-dimension converge to any desired value $d \in (1, 2]$ by choosing Y so that it grows according to $X^{d/2}$. This provides us with a means of choosing our ζ-dimension to an arbitrary precision: simply choose Y so that its size is proportional to $X^{d/2}$ and increase the value of X until your ζ-dimension is within the desired tolerance of d.

To see this, assume that $Y = X^{d/2}$ and note that this means that the dominant factor in the limit of A will be $16Y^2 = 16X^d$ and the dominant factor in the limit of S will be $2X$. Therefore

$$\begin{aligned}
\lim_{X\to\infty}\left(\frac{\log_2(A)}{\log_2(S)}\right) &= \lim_{X\to\infty}\left(\frac{\log_2(16X^d)}{\log_2(2X)}\right) \\
&= \lim_{X\to\infty}\left(\frac{4 + \log_2(X^d)}{1 + \log_2(X)}\right) \\
&= \lim_{X\to\infty}\left(\frac{4 + d\log_2(X)}{1 + \log_2(X)}\right) \\
&= d
\end{aligned}$$

For a fixed X, the maximum difference ϵ between the target ζ-dimension d and the ζ-dimension of the construction is therefore on the order

$$\epsilon = O\left(\frac{4 + d\log_2(X)}{1 + \log_2(X)} - d\right)$$

$$\epsilon = O\left(\frac{4 + d\log_2(X) - d - d\log_2(X)}{1 + \log_2(X)}\right)$$

$$\epsilon = O\left(\frac{4 - d}{1 + \log_2(X)}\right)$$

We therefore find that $X = O(2^{d/\epsilon})$ and noting that X is just under (less a logarithmic term) half the side length of the generator G , we find that the generator, likewise, has a side length of $O(2^{d/\epsilon})$.

10 A Self-Describing Embedded Circuit for the Sierpinski Cacarpet

Our second fractal construction relies on the self-describing circuit technique of Section 7.

THEOREM 10.1. *There is an evaluable circuit $C_\square$ with domain K^∞ and with an empty input bus which is self-describing. Moreover, $C_\square$ has only one gate without inputs.*

The remainder of this section is the description of $C_\square$. This circuit is built from two sets of messages: layer 1 messages in Σ_1 and layer 2 messages in Σ_2, and three fractal structures: a wiring layer $W_\square : K^\infty \to \mathcal{W}$, and two function layers, F_1 operating on Σ_1 and F_2 operating on Σ_2.

The wiring layer $C_w : K^\infty \to \mathcal{W}$ is the fixed point of a substitution $\kappa_w : \mathcal{W} \to \mathcal{W}^K$ starting from a seed wiring s_w:

$$\begin{cases} C_w(\vec{0}) = s_w \\ C_w(\vec{z}) = \kappa_w(C_w\lfloor\frac{z}{K}\rfloor)(z \bmod K) \end{cases}$$

The definition of the function layers is a bit more indirect: there are two sets of labels L_1 and L_2 respectively, with two rules $\kappa_1 : \mathcal{W} \to L_1^K$ and $\kappa_2 : \mathcal{W} \times L_2 \to L_2^K$. The rule κ_2 takes the form of a substitution.

Their fixpoints, starting from two seed labels $\mathfrak{s}_1$ and $\mathfrak{s}_2$ define two tiling of K^∞ with labels of L_1 and L_2 respectively: Λ_1 and Λ_2. On layer 1, Λ_1 is defined for each position according to the wiring of its parent:

$$\Lambda_1(\vec{z}) = \kappa_1(C_w(\lfloor\frac{z}{K}\rfloor))(z \bmod K)$$

On layer 2, Λ_2 is defined for each position according to the wiring *and label* of its parent in a substitutive manner:

$$\begin{cases} \Lambda_2(\vec{0}) = \mathfrak{s}_2 \\ \Lambda_2(\vec{z}) = \kappa_2(C_w(\lfloor\frac{z}{K}\rfloor), C_2(\lfloor\frac{z}{K}\rfloor))(z \bmod K) \end{cases}.$$

Finally, the actual gate functions are defined from two functions, instantiate$_1$ on layer 1 and instantiate$_2$ on layer 2, which define a function on Σ_1 (respectively Σ_2) from four elements, two wirings and two labels in L_1 (respectively L_2), one each for the gate and its parent. Together with the substitutions κ_i, they define a circuit $\mu_i(w, l)$ with domain K known as the layer-i meta-gate obtained by w and l:

$$\begin{cases} \mu_1(w,l) : K \to \text{Gates}(\Sigma_1) \\ \mu_1(w,l) : \vec{z} \mapsto \{\text{ wiring} : \kappa_w(w)(\vec{z}),\ \text{func} : \text{instantiate}_1(w, l, \kappa_w(w)(\vec{z}), \kappa_1(w)(\vec{z}))\} \\ \mu_2(w,l) : K \to \text{Gates}(\Sigma_2) \\ \mu_2(w,l) : \vec{z} \mapsto \{\text{ wiring} : \kappa_w(w)(\vec{z}),\ \text{func} : \text{instantiate}_2(w, l, \kappa_w(w)(\vec{z}), \kappa_2(w,l)(\vec{z}))\}. \end{cases}$$

In this circuit, the wiring of each gate is given by $\kappa_w(w)$, and its function is given by instantiating the label given by κ_i. When iterating this process, a gate g appearing at position $\vec{z}$ in some parent meta-gate $\mu_i(l, w)$, the child meta-gate $\mu_i(g)$ associated with g is defined as $\mu_i(g\cdot\text{wiring}, \kappa_i(l, w))$. The functions instantiate$_i$ are each injective in their last argument, ensuring this does not create any ambiguity.

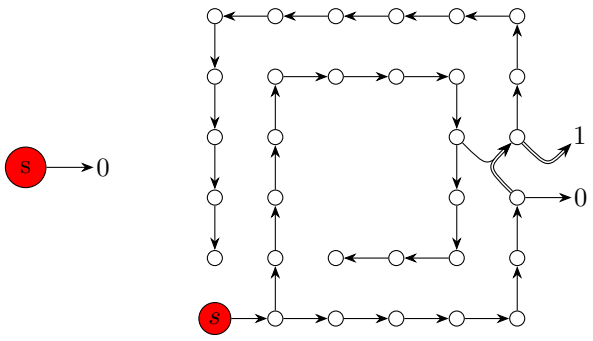


Figure 36: $\kappa_1(w_s, \mathfrak{seed})$, the first iteration of κ_1 on the seed gate. The wirings are given graphically; all gates have the $\mathfrak{normal}$ label, except the s at $(0,0)$ which has label $\mathfrak{seed}$. For the wiring with two inputs, the fat arrow represents the input number 0; all output wires have number 0: $w\cdot\text{output-num}(d) = 0$

Finally, these meta-gates beget the two layers of circuits C_1 and C_2 by:

$$\begin{cases} C_i(\vec{0}) = \{\text{ wiring} : s_w,\ \text{func} : s_i\} \\ C_i(\vec{z}) = \mu_i(C_i(\lfloor \frac{\vec{z}}{K} \rfloor))(\vec{z} \bmod K) \end{cases}.$$

By this definition, the wirings of $C_1(\vec{z})$ and $C_2(\vec{z})$ are the same —they are given by C_w, so $C_\square : \vec{z} \mapsto C_1(\vec{z}) \otimes C_2(\vec{z})$ defines a circuit with alphabet $\Sigma_1 \times \Sigma_2$ and domain K^∞.

The next subsections are the description of C_w, followed by those of C_1, then C_2, and finally the proof of that $C_\square$ is self-describing.

10.1 The wiring layer The seed wiring s_w is represented on the left of Figure 36. On the right of the same figure is its image $\kappa_w(s_z)$.

For any wiring w, $\kappa_1(w)$ will only contain wirings with at most two inputs sides, and these input sides are adjacent. Thus, all gates in C_w have at most two input sides and they are adjacent. Hence, κ_w only needs to be defined on wirings with that property.

The definition of κ_w is isotropic: κ_w commutes with a rotation of $\pi/2$ and with reflections. This property will be upheld simply by giving the definition of κ_w *up to rotation and reflection*.

Lastly, for each input wire in w there are two wires in the input bus of $\kappa_w(w)$, and for each output wire in w there are two wires in the output bus of $\kappa_w(w)$. The position of these wires are as follows, up to rotation:

input of w		input i of $\kappa_w(w)$		
$w\cdot$ Inputs	direction	position	direction	$i\cdot$ Inputs
(W)	W	$(0,2)$	W	(W)
		$(0,3)$	W	(S,W)
(S,W)	W	$(0,0)$	W	(S,W)
		$(0,1)$	W	(S,W)
	S	$(0,0)$	S	(S,W)
		$(1,0)$	S	(S,W)

output of g		output o of $\kappa_w(g)$		
$w\cdot\text{outwires}(d)$	direction d	position	direction d'	$i\cdot\text{outwires}(d')$
(W)	E	$(5,2)$	E	W
		$(5,3)$	E	(S,W)
(S,W)	E	$(5,0)$	E	(S,W)
		$(5,1)$	E	(S,W)
(S,W)	N	$(0,5)$	N	(S,W)
		$(1,5)$	N	(S,W)

Thus, if w has inputs $w\cdot\text{Inputs} = (W)$, $w\cdot\text{Outputs} = \{N,E,S\}$, outputs $w\cdot\text{outwires}(N) = Sw$, $w\cdot\text{outwires}(E) = sW$ and $w\cdot\text{outwires}(S) = N$, then $\kappa_w(w)$ has the corresponding input and output busses

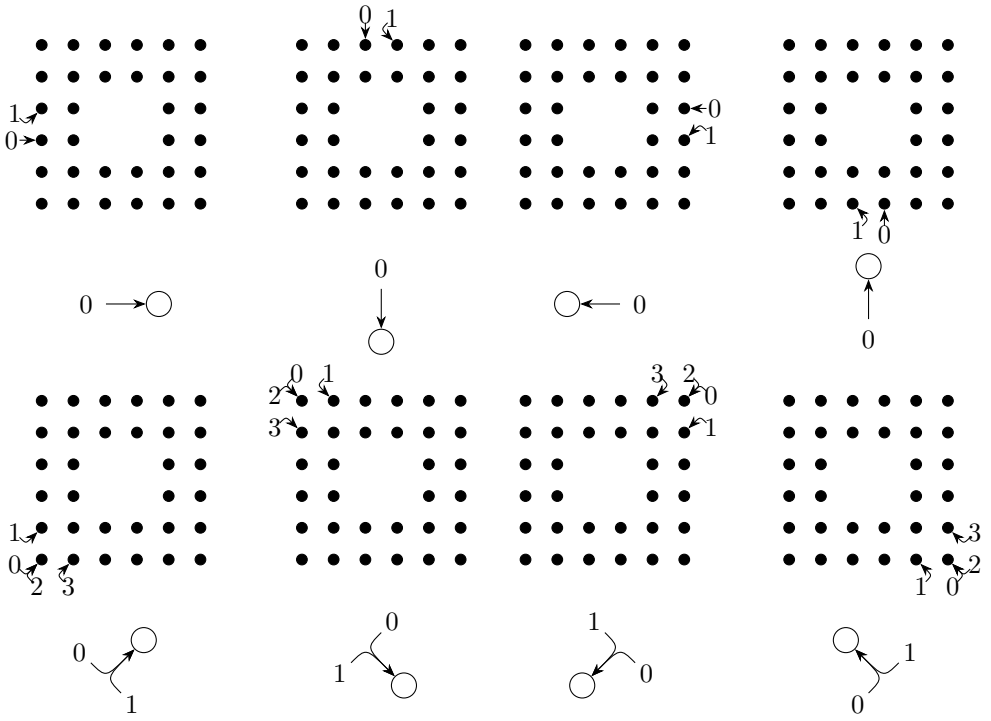


Figure 37: Location of the input wires in $\kappa_1(g)$ according to the input sides of $w = \text{wiring}\, g$. The output wires mirror the input sides of the neighboring tiles.

and wires:

$$\begin{cases} \vec{I} = \begin{pmatrix} (-1,2) \to (0,2)(\text{with wire W}) \\ (-1,3) \to (0,3)(\text{with wire Ws})) \end{pmatrix} \\ \vec{O} = \begin{pmatrix} (0,5) \to (0,6)(\text{with wire Sw}) \\ (1,5) \to (0,6)(\text{with wire Sw}) \\ (5,0) \to (6,0)(\text{with wire Ws}) \\ (5,1) \to (6,1)(\text{with wire Ws}) \\ (2,0) \to (2,-1)(\text{with wire N}) \\ (3,0) \to (3,-1)(\text{with wire Nw})) \end{pmatrix} \end{cases}$$

Wirings with no inputs All wirings without input are sent by κ_w to the array of wirings represented on Figure 36. The seed wiring s_w is the only one to effectively appear when iterating κ_w.

Wirings with one input A wiring with one input is cut according to Figure 38.

The wirings in $\kappa_w(g)$ depend on the input and outputs of w. Since w has only one input, it is sufficient, up to rotation, to examine the case where $w\cdot\text{Inputs} = \{W\}$. The black arrows of Figure 38 depict the case where $w\cdot\text{Outputs} = \varnothing$. For each $d \in w\cdot\text{Outputs}$, some extra wires are needed. The additional wires for an output in the N direction depend on whether the directions appearing in $w\cdot\text{outwires}(N)$ are $\{S\}$, $\{S, E\}$ or $\{S, W\}$. The wires for the other output directions are derived from these by rotation. Figure 38 has one direction with each case, showing the complete range of possibilities for the extra wires; they are represented in dotted lines on the figure.

Wirings with two (adjacent) inputs The image of such a wiring w by κ_w is defined on Section 10.1, which is rotated and reflected so that the S side of the figure is mapped to the 0 input of w, and the W side to the 1 input of w.

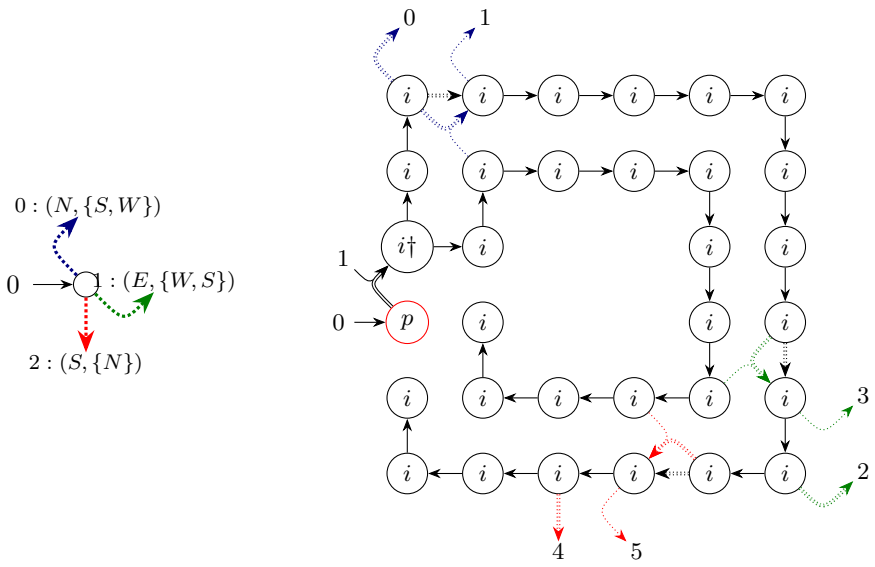


Figure 38: $\kappa_1(w, l)$ when $w \cdot \text{Inputs} = (W)$. The wirings are given graphically; on layer 1, all gates have the **normal** label (black), except for the gate at $(0, 3)$ which has label **input** (red). Dotted segments are parts of the wiring only if the corresponding element is in the outputs of w. For wirings with two inputs, the fat arrow represents the first input. The † marks the potential position for label $\mathfrak{dA}$ on layer 2.

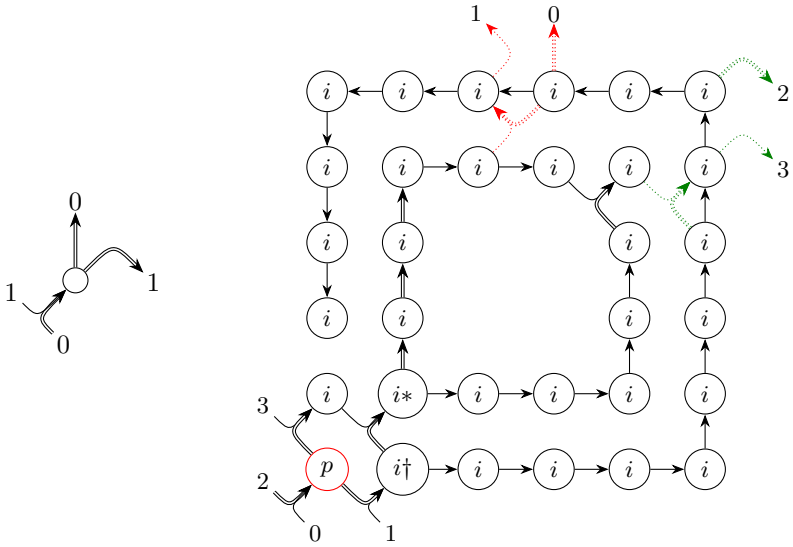


Figure 39: $\kappa_1(w, l)$ when $\text{Inputs}\, w = (S, W)$. The wirings are given graphically; all gates have the **normal** label except for the input gate at $(0, 0)$ which has label **input**. This configuration gets rotated and reflected according to the inputs of the gate, so that the arrows on the outer ring form a path from the side with input 0 to the side with input 1, "the long way around". The '*' marks the *special* position on layer 2, and the † marks the position where label $\mathfrak{dA}$ may appear.

LEMMA 10.1. *For each gate w, $\kappa_w(w)$ is the wiring of a normal circuit. Moreover, C_w is the wiring of a normal, closed circuit.*

Proof. This follows from observation of the schemata describing each $\kappa_w(w)$ and observing that for two wirings w, w', if an output side in direction d of w matches an input side in direction $-d$ of w', then the corresponding sides of $\kappa_w(w)$ and $\kappa_w(w')$ also match.

Additionally, each iteration of κ_w on s_w is without inputs, so C_w is closed. ∎

10.2 Layer 1 The components of layer 1 are an alphabet Σ_1, a finite set of labels L_1, a substitution $\kappa_1 : \mathcal{W} \mapsto L_1^K$, and for each label $l \in L_1$, and an instantiation function instantiate$_1$. Layer 1 can then be realized as described above into a circuit with the wirings of C_w and the functions given by instantiating the fixpoint of κ_1.

The set of labels is $L_1 = \{\mathfrak{s}_1, \mathfrak{normal}, \mathfrak{input}\}$.

The elements of Σ_1 are *layer 1 messages*.

DEFINITION 10.1. (LAYER 1 MESSAGE) *A layer 1 message is a triple* $\{\,\text{pos} = \vec{z} \in K_1, \text{parent-wiring} = w \in \mathcal{W}, \cdot\,\text{parent} = l \in L_1\}$.

The circuit C_1 is going to be defined by iterating κ_w and κ_1 starting from s_w and $\mathfrak{s}_1$, then instantiating the labels.

Labels The label function $\kappa_1 : \mathcal{W} \to L_1^K$ assigns the label $\mathfrak{normal}$ at all positions except:

- $\kappa_1(s_w)(\vec{0}) = \mathfrak{s}_1$
- if w has at least one input, then $\kappa_1(w)(\vec{z}) = \mathfrak{input}$ for the position $\vec{z}$ which receives the input wire 0 in $\kappa_w(w)$.

The seed function The starting label is $\mathfrak{s}_1$, it only ever appears at position $(0,0)$ in $\kappa_1(s_w, \mathfrak{s}_1)$.

Since s_1 has no inputs, its associated function $f_s = \text{instantiate}_1(s_w, \mathfrak{s}_1, s_w, \mathfrak{s}_1)$ is a constant function with value $f_s() = \{\,\text{pos} : (0,0), \text{parent-label} : \mathfrak{s}_1, \text{parent-wiring} : s_w\}$.

Gate functions for layer 1 There are two types of functions for the gates output by $\kappa_1(g)$ other than the seed. They have a either an *increment* function incr$[k, D]$ with $k \in \{1, 2\}$ and $D \in \{N, E, S, W\}$ or a *reparenting* function set-parent$[k, D, w, c]$, with $k \in \{1, 2\}$, $D \in \{N, E, S, W\}$, w a wiring and $c \in C$ a color. The versions with $k = 2$ take two inputs but ignore the second one. The functions incr and set-parent are defined as follows:

$$\begin{aligned}
\text{incr}[1, D](m) \cdot \text{parent} &= m \cdot \text{parent}, \\
\text{incr}[1, D](m) \cdot \text{pos} &= \text{pos}\, m + D \\
\text{set-parent}[1, D, w, c](m) \cdot \text{parent} &= (w, c) \\
\text{set-parent}[1, D, w, c](m) \cdot \text{pos} &= m \cdot \text{pos} + D \\
\text{incr}[2, D](m, m') &= \text{incr}[1, D](m) \\
\text{set-parent}[2, D, w, c](m, m') &= \text{set-parent}[1, D, w, c](m).
\end{aligned}$$

The function of each gate is fixed from its label and wiring, and those of its parent through instantiate$_1$ as follows:

$$\begin{cases} \text{instantiate}_1(w_p, l_p, w, \mathfrak{normal}) = \text{incr}[k, D] \\ \text{instantiate}_1(w_p, l_p, w, \mathfrak{input}) = \text{set-parent}[k, D, w_p, l_p], \end{cases}$$

where k is the number of inputs of w, and D is the direction of its first input.

Behavior and Self-Description of Layer 1 The circuit C_1 obtained from κ_1 is normal, by Lemma 10.1. Let $e = \widetilde{C_1} : K^\infty \times \{N, E, S, W\} \to \Sigma_l$ be the evaluation function of C_1. That function e enjoys a simple description, which reflects the fact that in C_1, the different meta-gates do not actually communicate. On layer 2 however, there will be some communication between meta-gates, as described in the next section.

LEMMA 10.2. *Let $w \in \mathcal{W}$, $l \in L_1$ and $a : p \mapsto p'$ be an internal or outgoing arc in $\kappa_w(w)$. If w has no inputs, assume $l = \mathfrak{s}_1$.*

For any input $\vec{\imath}$ of $\mu_1(w,l)$, let $m = \widetilde{\mu_1(w,l)}(\vec{\imath}, a)$ be the value of arc a on input $\vec{\imath}$. Then $m{\cdot}\,\mathrm{pos} = p$, $m{\cdot}$ parent-wiring $= w$ and $m{\cdot}$ parent-label $= l$.

Proof. By induction on the non-incoming arcs of the (acyclic) dependency graph D of $\kappa_w(w)$.

If w is the seed wiring s_w, then the root of D is also s_1 and its outputs satisfy $e_g(a){\cdot}\,\mathrm{pos} = (0,0)$, $e_g(a){\cdot}$ parent-wiring $= s_1{\cdot}$ wiring and $e_g(a){\cdot}$ parent-label $= \mathfrak{s}_1$.

Otherwise, $\kappa_1(w)$ has a unique position $\vec{z_i}$ with label $\mathtt{input}$, corresponding to a gate in $\mu_1(l,w)$ with function set-parent$[k, D, w, l]$, for some k and D. By definition of set-parent, its outputs satisfy $e_g(a){\cdot}\,\mathrm{pos} = \vec{z_i}$, $e_g(a){\cdot}$ parent-wiring $= w$ and $e_g(a){\cdot}$ parent-label $= l$.

In both cases, each other gate g' at position $\vec{z}$ of $\mu_1(l,w)$ has function incr$[k, D]$, where k is the number of inputs of g', and D is the direction of its first input arc $i = \vec{z} - D \overset{D}{\mapsto} z$. By induction, $e_g(i){\cdot}\,\mathrm{pos} = z - D$ and $e_g(i)\cdot$ parent $= (g{\cdot}$ wiring, label$(g))$. By definition of incr$[k, D]$, each of the output arcs o of g' verify $e_g(o){\cdot}\,\mathrm{pos} = (\vec{z} - D) + D = \vec{z}$ and $e_g(a){\cdot}$ parent-wiring $= w$ and $e_g(a){\cdot}$ parent-label $= l$. ∎

Additionally, C_1 is "mostly self-describing": the first input of a gate g is enough to recover g, except for the value of w and c in gates with a function of the form set-parent$[k, D, w, c]$.

LEMMA 10.3. *For a position $p \in K^\infty$, let $\vec{e} =$ Inputs $C_1(p){\cdot}$ wiring be the input arcs of $C_1(p)$; let d_i be the direction of e_i.*

There is a function dec-gate $: \Sigma_1 \times \{N, E, S, W\} \to \mathrm{Gates}(\Sigma_1) \cup \bot$ *such that for all $p \in K^\infty$,*

- *if $C_1(p){\cdot}$ func is* incr$[k, d_0]$, *then* dec-gate$(\widetilde{C_1}(e_0), d_0) = C_1(p)$
- *if $C_1(p){\cdot}$ func is* set-parent$[k, d_0, -, -]$ *then* dec-gate$(\widetilde{C_1}(e_0), d_0) = \bot$.

Proof. The function dec-gate is defined as follows: let $m \in \Sigma_1$ be a local message, and $d \in \{N, E, S, W\}$. Let $p = m{\cdot}\,\mathrm{pos}$, if $p - d \notin \{0, \ldots, 5\}^2$, then dec-gate$(m, d) = \bot$. Otherwise, dec-gate$(m, d)$ is the gate at position p in $\mu_1(m{\cdot}$ parent-wiring$, m \cdot$ parent$)$.

By Lemma 10.2, dec-gate satisfies the lemma. ∎

Moreover, when dec-gate$(e(e_O), d_0) = \bot$, g itself cannot be determined, but its label and wiring can, as well as $\mu_1(g)$.

COROLLARY 10.1. *For a position $p \in K^\infty$, let $\vec{e} = C_1(p){\cdot}$ Inputs $\cdot$ wiring be the input arcs of $C_1(p)$; let d_i be the direction of e_i,*

- *there is a function* dec-gate$_w$ $: \Sigma_1 \times \{N, E, S, W\} \to \mathcal{W}$ *such that for each position $p \in K^\infty$,* dec-gate$_w(\widetilde{C_1}(e_0), d_0) = C_1(p){\cdot}$ wiring
- *there is a function* dec-gate$_l$ $: \Sigma_1 \times \{N, E, S, W\} \to \mathcal{W}$ *such that for each position $p \in K^\infty$,* dec-gate$_l(\widetilde{C_1}(e_0), d_0) = \Lambda_1(p)$

Proof. For dec-gate$_w$ and dec-gate$_l$, it suffices to observe that whenever dec-gate$(\widetilde{C_1}(e_0), d_0) = \bot$ at some position p, $\widetilde{C_1}(p)$ has label $\mathtt{input}$, and its wiring only depends on its position within its metagate. ∎

10.3 Layer 2 A second layer is needed in order to get full self-description of the circuit on K^∞. This layer routes global information between the meta-gates so that the input gate of each meta-gate can be indentified by its incoming global message. The construction needs to "tie the knot", so it is not only needed to recover the identity of the input gate on the layer 1, but also on layer 2 itself.

This layer is defined by a set L_2 of labels, an alphabet Σ_2, the label substitution κ_2 and its instantiation function. In contrast with layer 1, κ_2 takes as input a wiring, as well as a label. This makes the definition of Λ_2 recursive. In contrast with layer 1, on layer 2, the two-step definition of gates using labels and the function instantiate is actually *needed* because of a subtelty related to the tying of the knot. The functions of some gates make use of κ_2. Thus κ_2 shall not directly manipulate the gates or their function, lest the definition of layer 2 becomes cyclical and possibly ill-founded.

The set of layer 2 labels is $L_2 = \{\mathfrak{s}, \mathfrak{i}_1, \mathfrak{i}_2, \mathfrak{dL}, \mathfrak{dA}\}$.

Layer 2 messages A message on layer 2 is an element of Σ_2; it is built from:

- $m\cdot$ parent-label$_2$ $\in L_2$,
- $m\cdot$ global, itself consisting of:
  - $m\cdot$ global$\cdot$ label$_2$ $\in L_2$.
  - $m\cdot$ global$\cdot$ ancestor-msg $\in \Sigma_1$

These messages make C_2 self-describing by completing the information available in layer 1 and used in Lemma 10.3. A message m_2 output by a gate g_2 in $\kappa_2(p)$ identifies p through $m_2\cdot$ parent-label$_2$ if g_2 is not the input gate of $\kappa_2(p)$, as in Lemma 10.3. The global part $m_2\cdot$ global identifies *some ancestor* of g_2, on layer 1 through $m\cdot$ global$\cdot$ ancestor-msg and on layer 2 through $m\cdot$ global$\cdot$ label$_2$. This global information will enable the determination of the entry gate of each meta-gate, on both layers. From two messages $m_l, m_g \in \Sigma_2$, two messages $(m_1, m_2) = \text{extract}(m_l, m_g) \in \Sigma_1 \times \Sigma_2$ can be extracted by reading the local information from m_l, and the global information from m_g, as follows:

$$\begin{cases} m_1 = m_l\cdot \text{global}\cdot \text{ancestor-msg} \\ m_2\cdot \text{parent-label}_2 = m_l\cdot \text{global}\cdot \text{label}_2 \\ m_2\cdot \text{global} = m_g\cdot \text{global}\,. \end{cases}$$

The converse operation, embedding, takes as input three messages, a payload $(p_1, p_2) \in (\Sigma_1 \times \Sigma_2)$ and a context $c \in \Sigma_2$, and yields two messages m_l and m_g

$$\begin{cases} m_l\cdot \text{parent-label}_2 = m_g\cdot \text{parent-label}_2 = c\cdot \text{parent-label}_2 \\ m_l\cdot \text{global}\cdot \text{label}_2 = p_2\cdot \text{parent-label}_2\,. \\ m_l\cdot \text{global}\cdot \text{ancestor-msg} = p_1 \\ m_g\cdot \text{global} = p_2\cdot \text{global}\,. \end{cases}$$

Extracting and embedding are dual operations, in the sense that

$$\forall c, p_1, p_2, \text{extract}\circ\text{embed}(c, p_1, p_2) = (p_1, p_2).$$

The substitution κ_2 Given a wiring w and a label $l \in L_2$, the substitution κ_2 yields a label for each position in K;

- $\mathfrak{s}_2$ for the seed gate, i.e. for position $(0, 0)$ if w has no inputs;
- $\mathfrak{dL}$ for the gate receiving the input number 0 of the meta-gate;
- $\mathfrak{dA}$ for the gate receiving the input number 1 of the meta-gate whenever $l = \mathfrak{dL}$;
- when w has two inputs, there is a *special* position within the meta-gate where the value depends on l as follows:
  - $\mathfrak{i}_2$ if $l \in \{\mathfrak{i}_2, \mathfrak{dL}\}$,
  - $\mathfrak{dA}$ if $l = \mathfrak{dA}$;
- otherwise, $\mathfrak{i}_1$ for gates with one input and $\mathfrak{i}_2$ for gates with two inputs.

The special position is marked on Figure 38 and Section 10.1 by a star.

The layer-2 seed gate The function $f_s^2 = \text{instantiate}_2(s_w, \mathfrak{s}_2, s_w, \mathfrak{s}_2)$ of the seed gate is a constant function with value $f_s^2() = \{\,\text{parent-label}_2 : \mathfrak{s}_2,\ \text{global} : \{\,\text{label}_2 : \mathfrak{s}_2,\ \text{ancestor-msg} : f_s^1()\}\}$. Recall that $f_s^1()$ is the message output by the seed gate on layer 1.

Gate functions for layer 2 The functions of the gates depend on their label and on the direction D of their first input, as dictated by the function instantiate:

$$\begin{cases} \text{instantiate}(D, \mathfrak{s}) = f_s^2 \\ \text{instantiate}(D, \mathfrak{i}_1) : x \mapsto x \\ \text{instantiate}(D, \mathfrak{i}_2) : (x, y) \mapsto (x, y) \\ \text{instantiate}(D, \mathfrak{d}\mathfrak{L}) = \text{decode}_L[D] \circ \text{incr}_G[D] \\ \text{instantiate}(D, \mathfrak{d}\mathfrak{A}) = (m_l, m_g) \mapsto \text{decode}_A[D](\text{incr}_G[D](m_l), m_g), \end{cases}$$

where D is the direction of the first input of w_p.

The gate with labels $\mathfrak{i}_1$ or $\mathfrak{i}_2$ are wires; their function is the identity function of arity 1 or 2 respectively.

Given $m \in \Sigma_2$, the increment functions $\text{incr}_G[D]$ increments ancestor-msg global $m\cdot$ pos by the unit vector of direction D.

The two decoding functions decode_L and decode_A are based on the same underlying function $\text{decode} : \{N, E, S, W\} \times \Sigma_1 \times \Sigma_2 \to \Sigma_1 \times \Sigma_2$ defined as follows: let $m_1 \in \Sigma_1, m_2 \in \Sigma_2$, pose $z = m_1\cdot \text{pos}$, $z_a = m_2\cdot \text{global}\cdot \text{ancestor-msg}\cdot \text{pos}$, $a = m_2\cdot \text{global}\cdot \text{ancestor-msg}$. Let l_p be defined as $\kappa_2(a\cdot \text{parent-wiring}, a\cdot \text{parent-label}, m_2\cdot \text{global}\cdot \text{label}_2, p')$ if $z \in K_1$, and $\mathfrak{d}\mathfrak{L}$ otherwise. Then $\text{decode}[D](m_1, m_2)$ is the pair (m_1', m_2') with:

$$m_1' = \left\{ \begin{array}{ccc} \text{pos} & : & z \bmod K \\ \text{parent-wiring} & : & \text{dec-gate}_w(a, D) \\ \text{parent-label} & : & \text{dec-gate}_l(a, D) \} \end{array} \right\}$$

$$m_2' = \left\{ \begin{array}{ccc} \text{parent-label}_2 & : & p \\ \text{global}\cdot \text{label}_2 & : & m\cdot \text{label}_2 \cdot \text{global} \\ \text{global}\cdot \text{ancestor-msg} & : & \left\{ \begin{array}{ccc} \text{parent-wiring} & : & m\cdot \text{global}\cdot \text{ancestor-msg}\cdot \text{parent-wiring} \\ \text{parent-label} & : & m\cdot \text{global}\cdot \text{ancestor-msg}\cdot \text{parent-label} \\ \text{pos} & : & z_a \bmod K_1 \end{array} \right\} \end{array} \right\}$$

For a direction D and a message $m_2 \in \Sigma_2$, take an arbitrary $m_1 \in \Sigma_1$ and let $(m_1', m_2') = \text{decode}[D](m_1, m_2)$; the value of m_2' does not depend on m_1, so $\text{decode}_L[D](m_2)$ is defined to be the $m_2' \in \Sigma$ returned by $\text{decode}(m_1, m_2)$ for any m_1.

For a direction D and $m_l, m_g \in \Sigma_2$, $\text{decode}_A[D](m_l, m_g)$ is defined as follows. Let $(m_1, m_2) = \text{extract}(m_l, m_g)$, $(m_1', m_2') = \text{decode}[D](m_1, m_2)$. Then $\text{decode}_A[D](m_l, m_g)$ is the pair m_l', m_g' with:

$$\begin{cases} m_l'\cdot \text{parent-label}_2 = m_g'\cdot \text{parent-label}_2 = m_l\cdot \text{parent-label}_2 \\ m_l'\cdot \text{global}\cdot \text{ancestor-msg} = m_1' \\ m_l'\cdot \text{global}\cdot \text{label}_2 = m_2'\cdot \text{parent-label}_2 \\ \text{global}\, m_g' = m'\cdot \text{global} \end{cases}$$

The function decode_A is engineered in order to enjoy the following property, a kind of commutation between decode and extract.

LEMMA 10.4. *For any direction $D \in \{N, E, S, W\}$,*

$$\text{decode}[D] \circ \text{extract} = \text{extract} \circ \text{decode}_A[D]$$

Proof. By computation. ∎

Properties of C_2 The messages passing through the circuit C_2 built above hold all the necessary information for $C_\square$ to be self-describing: in other words, C_2 carries all the information needed to determine the entry gate of each meta-gate in C_1, as well as the information needed to determine each of its gates.

The global $\cdot$ ancestor-msg part of the messages on input 0 each meta-gate simulate the gates of layer 1, as long as each meta-gate simulating a gate with label $\mathfrak{input}$ receives on input 1 the message of its parent meta-gate.

LEMMA 10.5. *let $w \in \mathcal{W}$ with k inputs, $l_2 \in L_2$, and M be the circuit $\mu_2(w, l_2)$. Note that M has $2k$ inputs. Let D be the direction of the first input of w.*

Let $\vec{\imath} \in \Sigma^{2k}$, and let $\vec{o}$ be the output of M on input $\vec{\imath}$. Pose $i_a = i_0 \cdot$ ancestor-msg $\cdot$ global and $i_p = i_1 \cdot$ ancestor-msg $\cdot$ global.

Then, if $g_1 =$ dec-gate$(i_a, D) \neq \bot$, then o_0 is the output of g_1 on input i_a; otherwise, if dec-gate$(i_a, D) = \bot$, *then o_0 is the output of* dec-gate$_c(i_p, D)(i_a \cdot \text{pos})$ *on input i_a.*

Proof. By computation. ∎

The rest of the global part of the messages allows C_2 to simulate itself.

LEMMA 10.6. *Let $D \in \{N, E, S, W\}$, $w \in \mathcal{W}$ with one input in direction D, $l_2 \in L_2$. Let $f =$* instantiate(D, l_2) *and $C = \mu_2(w, l_2)$.*

Let $c \in \Sigma_1, i_1 \in \Sigma_1, i_2 \in \Sigma_2$ and $(m_l, m_g) =$ embed(c, i_1, i_2). *Let (o_0, o_1) be the outputs of C on input (m_l, m_g). Then* extract$(o_0, o_1) = f(i_2)$.

Proof. The proof proceeds by case on l_2, which can be either $\mathfrak{i}_1$ or $\mathfrak{dL}$. If $l_2 = \mathfrak{i}_1$, then f is the identity function, and it suffices to follow the wirings to check the result.

If $l_2 = \mathfrak{dL}$, then following the wirings reduces the desired equality to the definition of decode$_L[D]$.
∎

LEMMA 10.7. *Let $w \in \mathcal{W}$ with two inputs, $l_2 \in L_2$. Let $f =$* instantiate(l_2) *and $C = \mu_2(w, l_2)$.*

For $k \in \{0, 1\}$, let $c^k \in \Sigma_1, i_1^k \in \Sigma_1, i_2^k \in \Sigma_2$ and $(m_l^k, m_g) =$ embed(c^k, i_1^k, i_2^k). *Let $(o_0^0, o_1^0, o_0^1, o_1^1)$ be the outputs of C on input $(m_l^0, m_g^0, m_l^1, m_g^1)$. Then for $k \in \{0, 1\}$* extract(o_0^k, o_1^k) *is the k-th component of $f(i_2^0, i_2^1)$.*

Proof. The proof proceeds by case on l_2. If $l_2 \in \{\mathfrak{s}, \mathfrak{i2}, \mathfrak{dA}\}$, following the wirings in $\kappa_2(w, l_1, l_2)$ confirms that the lemma holds.

If $l_2 = \mathfrak{dL}$, the lemma follows from Lemma 10.4 by again following the wirings. ∎

Together, these properties entail a substitutive structure of the messages in $C_\square$. Going up the hierarchy, the message between two gates of $C_\square$ can be extracted from the messages between the corresponding meta-gates.

LEMMA 10.8. *Let $a = p \to p'$ be a wire between two positions p, p' of K^∞ in direction D. Let a_l, a_g be the two wires crossing the edges between pK and $p'K$ in clockwise order looking in direction D (i.e., if D is E, a_l is the northernmost of the two; if D is S, the westernmost...).*

Let $m = \widetilde{C_\square}(a)$, $l = (l_1, l_2) = \widetilde{C_\square}(a_l)$ and $g = (g_1, g_2) = \widetilde{C_\square}(a_g)$.

Then extract$(l_2, g_2) = m$.

Proof. By induction on p, following the wires of $C_\square$. ∎

LEMMA 10.9. *Let $\vec{p} \in K^\infty$. Let $g_1 = C_1(\vec{p})$, $g_2 = C_2(\vec{p})$, $g_1' = C_1(\lfloor \frac{\vec{p}}{K} \rfloor)$, $g_2' = C_2(\lfloor \frac{\vec{p}}{K} \rfloor)$. Let $g = g_1 \otimes g_2 = C_\square(\vec{p})$, and $g' = g_1' \otimes g_2' = C_\square(\lfloor \frac{\vec{p}}{K} \rfloor)$. Let e be an output wire of g, and $m_1 \times m_2 = \widetilde{C_\square}(e)$ its value in $C_\square$.*

Then $m_2 \cdot$ parent-label$_2$ is the label of g_2', $m_1 \cdot$ parent-label is the label of g_1', $m_1 \cdot$ parent-wiring is $g' \cdot$ wiring, and $m_1 \cdot$ pos is $\vec{p} \bmod K$.

Proof. By the previous lemma, input 0 of each meta-gate $\mu_1(l_1, w) \otimes \mu_2(l_2, w)$ encodes l_1, l_2 and w. The gate after that input has label $\mathfrak{dL}$, so by definition of its function decode$_L$, its output satisfies the lemma. The other gates in the meta-gate preserve the local part of the messages. ∎

This local information is just what is needed to reconstruct each gate from its *output*, which is just short of self-description.

COROLLARY 10.2. *There are is a function* dec-gate$' : \Sigma_1 \times \Sigma_2 \to \mathcal{W} \times L_1 \times L_2 \times K$ *such that for any position $p \in K^\infty$ and any wire $a : p \to p'$ of $C_\square$,*

$$\text{dec-gate}'(\widetilde{C_\square}(a)) = (\Lambda_1(\lfloor p/K \rfloor), \Lambda_2(\lfloor p/K \rfloor), C_w(\lfloor p/K \rfloor), p \bmod K).$$

With a tiny bit of extra work, each gate g can be reconstructed from its input number 0, making $C_\square$ self-descriptive.

THEOREM 10.2. *The circuit $C_\square$ is self-descriptive.*

Proof. Each gate g in position p can be reconstructed from its set of input directions and the message on its input number 0.

If g has no inputs, then $g = s_1 \otimes s_2$.

Otherwise, let D be the direction of its first input wire, and $m = (m_1, m_2)$ the message coming into its first wire.

Let $(w, l_1, l_2, p') = \text{dec-gate}'(m)$. Note that $p' = (p - D) \bmod K$. If $p' + D$ belongs to K, then $\left\lfloor \frac{p}{K} \right\rfloor = \left\lfloor \frac{p-D}{K} \right\rfloor$, so p belongs to the same meta-gate as its predecessor $p - D$, hence $C_\square(p)$ is $\mu_1(w, l_1)(p' + D) \otimes \mu_2(w, l_2)(p' + D)$.

Otherwise, since $p' + D \notin K$, it must be the case that $\left\lfloor \frac{p}{K} \right\rfloor = \left\lfloor \frac{p-D}{K} \right\rfloor + D$: p and its predecessor $p - D$ are in neighboring meta-gates. Let $a = \left\lfloor \frac{p}{K} \right\rfloor$ be the position of the parent gate. The position $a \bmod k$ of a within its meta-gate is the position where input 0 enters that meta-gate.

A close examination of the values of $\mu_1(w, l_1)$ and $\mu_2(w, l_2)$ for all w, l_1, l_2 reveals that in each (non-seed) meta-gate the label and wiring of the gate in the position where input 0 comes only depends on the input directions of w. By Lemma 10.9, those directions are exactly those of $m_2 \cdot \text{global} \cdot \text{ancestor-msg} \cdot \text{parent-wiring} \cdot \text{outwires}(D)$. ∎

This concludes the proof of theorem 3.6. By theorem 3.3, this means that there is an aTAM system $S_\square$ which strictly self-assembles K^∞.

11 Admissible Generators for DSSF Strict Self-Assembly

One, two, infinity? We now want to characterize the generators G for which G^∞ is amenable to strict self-assembly.

11.1 Generalizing the Circuit Construction The circuit-based construction of Section 10 can be adaptated to any self-similar discrete fractal within which the communication pattern used by $C_\square$ can be embedded. Whether a particular fractal is amenable to hosting such a communication pattern depends on the ability of its generator G to transport information to copies of itself around it.

DEFINITION 11.1. *Let G be a finite subset of $\mathbb{N}^2$, the grid $G^\#$ is the subset of $\mathbb{Z}^2$ defined by:*

$$G^\# = \{\ p \in \mathbb{Z}^2 | p \bmod G \in G\}$$

The grid neighborhood graph G^+ of G is the subgraph of $G^\#$ induced by the distance 1 neighborhood of G:

$$G^+ = G \cup \{p \in G^\# | \exists d \in \{N, E, S, W\}, p + d \in G\}.$$

For a direction $d \in \{N, E, S, W\}$, the d-port in G^+ is $G^{+d} = \{p \in G^\# | p - d \in G\}$.

For $d, d' \in \{N, E, S, W\}$, the (d, d')-bandwidth of G is the number $G[d \leftrightarrow d']$ of vertex-disjoint paths from G^{+d} to $G^{+d'}$ in G^+.

In order to compare generators, the classical notions of *subgraph* and *graph subdivision* is useful, accounting for marked vertices.

DEFINITION 11.2. (POINTED SUBGRAPH) *Let G, H be two graph, each with marked vertices. The graph H is a* pointed subgraph *of G if H is a subgraph of G in such a way that any marked vertex of H is mapped to a marked vertex of G. The graph G may have extra marked vertices.*

DEFINITION 11.3. (EDGE SUBDIVISION) *Let $G = (V, E)$ be a graph, with marked vertices $(v_0, \ldots, v_{k-1}) \in V^k$. The edge subdivision operation for an edge $e = \{u, v\} \in E$ is the deletion of e from G and the addition of a new vertex $w \notin V$ and of the edges $\{u, w\}$ and $\{w, v\}$.*

This operation generates a new graph H, where the same vertices are marked as in G.

$$H = (V \cup \{w\}, (E \setminus \{u, v\}) \cap \{\{u, w\}, \{w, v\}\})$$

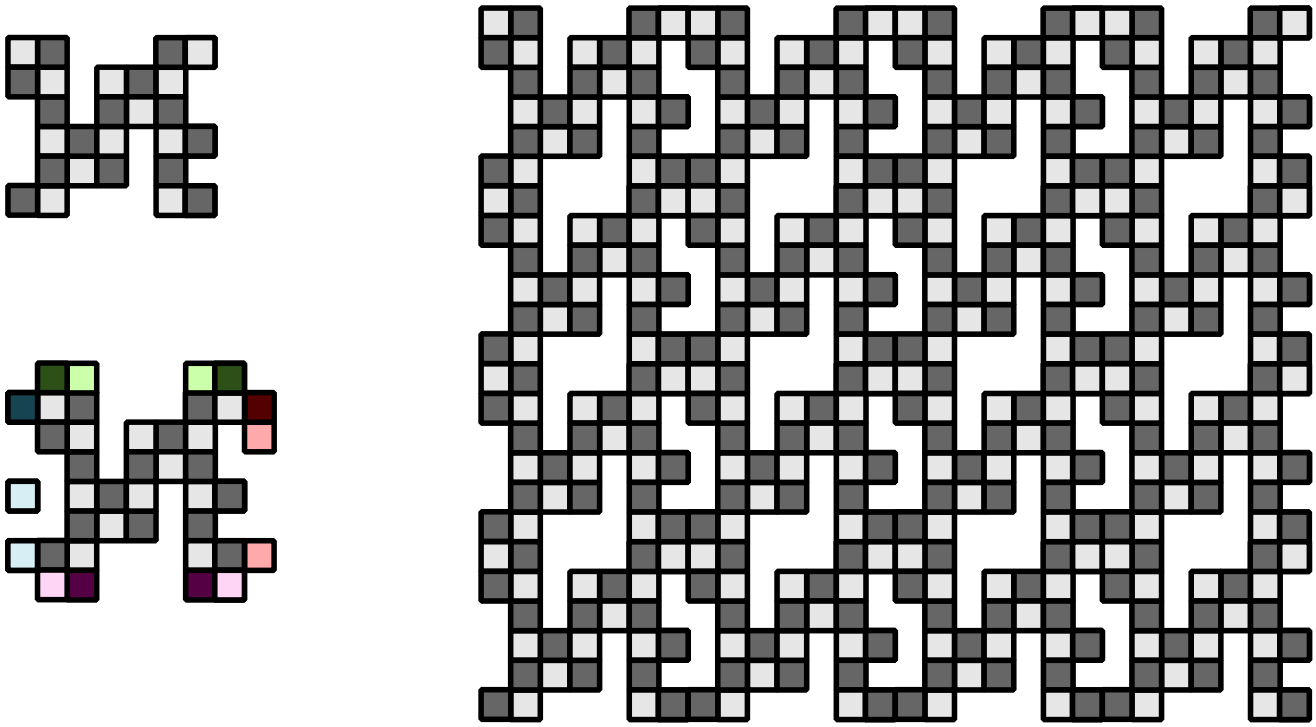


Figure 40: A finite shape G, its grid neighborhood G^+ and its grid graph $G^{\#}$.

Definition 11.4. (Graph Subdivision) *A graph with marked vertices which has been derived from G by a sequence of edge subdivision operations is called a pointed subdivision of G.*

Definition 11.5. (Subconnector) *Let G, H be finite, connected shapes of $\mathbb{N}^2$, with $(0,0) \in G \cap H$.*

Then H is a subconnector of G, written $H \preceq G$ if G^+ has a (pointed) subgraph which is a (pointed) subdivision of H^+.

The notion of subconnector is well-suited to the study of substitutions given the following properties.

Remark 2. *Let G, H, I be finite, connected shapes of $\mathbb{N}^2$, with $G \preceq H$, then:*

$$\sigma_I(G) \preceq \sigma_I(H)$$
$$\sigma_G(I) \preceq \sigma_H(I)$$

Lemma 3.1. *Let $G \ni (0,0)$ a finite, connected subgraph of $\mathbb{N}^2$ such that for some finite k, $K \preceq G^k$. Then there is a self-descriptive circuit with domain G^∞.*

Proof. First, notice that κ_w can be completed to cover the case where w has two opposite input directions, as represented on Figure 41 (modulo rotation and reflection).

Fix the vertices of G^+ which represent the vertices of K^+, the ones that sit in the middle of its edges, and the ones which are pending leaves.

Then it is possible to define a substitution $\gamma_w : \mathcal{W} \to \mathcal{W}^G$ by using κ_w to fix a subdivision of G^+, then adding the extra vertices to $\gamma_w(w)$. This process may create vertices in $\gamma_w(w)$ with two opposite inputs, for which the extra cases of Figure 41 are necessary.

The label substitutions κ_1 and κ_2 can also be adaptated to G^+, defining γ_1 and γ_2. In γ_1, all vertices which do not represent a vertex of K have label $\mathfrak{normal}$. For layer 2, each vertex of G^+ representing a vertex of K keeps its label, each new vertex gets a label $\mathfrak{i}_1$.

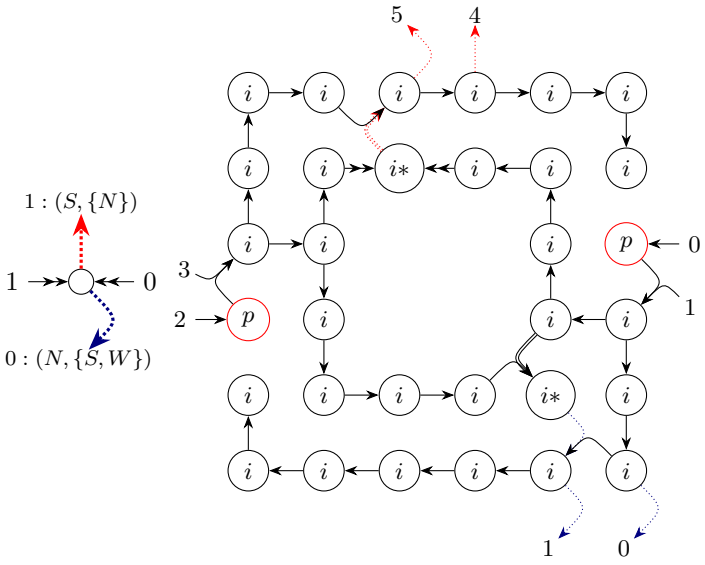


Figure 41: The value of $\kappa_w(w)$ when $w \cdot \text{Inputs} = \{E, W\}$. This case is not needed in Section 10, but it is necessary to generalize κ_w to G when K^+ is a subdivision of G^+. No position needs to be provisionned for $\mathfrak{dA}$, since both inputs must be in the same metagate.

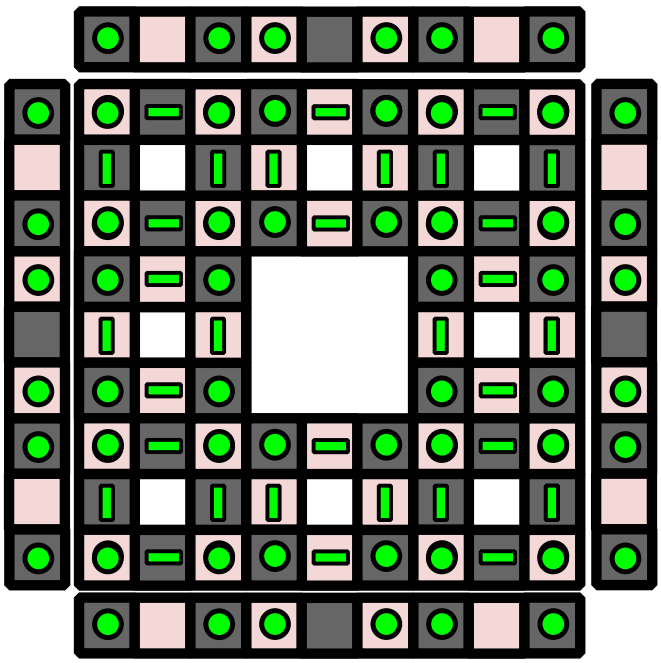


Figure 42: The second iteration S of the Sierpinski Carpet (in the center) is a subconnector of K: S^+ contains a subdivision of K^+, in which the cells with a circle correspond to vertices of K^+, and those with a bar are obtained by dividing the edge of K^+ the bar represents. Not all edges are subdivided: adjacent circles in G^+ are indeed adjacent in K^+.

Set $\Sigma_1^G = \Sigma_1$, except that pos takes values in G rather than K. Then Σ_2^G is Σ_2, except that ancestor-msg takes values in Σ_1^G rather than Σ_1.

Then a circuit C_G on $\Sigma_1^G \times \Sigma_2^G$ can be defined from γ_w, γ_1 and γ_2 like $C_\square$. That circuit C_G has the same properties as $C_\square$ and is also self-descriptive. When accounting for the local part of the messages, gates which are upstream from all the K-vertices show the local message of the corresponding input meta-gate.

Thus C_G is self-descriptive. ∎

The question is now which G are such that G^+ has a subdivision of K^+ as a subgraph. This condition seems constraining since K is a rather large graph with its 32 vertices. Yet, one can make any G larger by iterating the substitution generated by G before trying to self-assemble G^∞.

REMARK 3. *Let $G \ni (0,0)$ be a finite shape of $\mathbb{N}^2$. For any $k > 0$, $G^\infty = (G^k)^\infty$.*

By iterating σ_G, it is quite easy to get an instance of K as a subconnector, as long as one starts with sufficient vertical and horizontal bandwidth.

LEMMA 11.1. *Let $G \ni (0,0)$ be a finite shape of $\mathbb{N}^2$. If $G[N \leftrightarrow S] \geq 2$ and $G[E \to W] \geq 2$, then $K \preceq G^3$*

Proof. Let $H = \{0,1\}^2$. If $G[N \leftrightarrow S] \geq 2$ and $G[E \to W] \geq 2$, then $H \preceq G$: pick two disjoint North-South paths, two disjoint East-West paths, their four intersections can act as the vertices of H.

By Remark 2, since $H \preceq G$, $H^3 \preceq G^3$. But H^3 is an 8×8 grid, so $K \preceq H^3$.

Hence, $K \preceq G^3$. ∎

11.2 Limits to Strict Assembly of DSSFs in the ATAM model This characterization of generators for which the associated fractal can be strictly self-assembled is tight, by a generalization of the impossibility result of Hendricks et al. [25].

LEMMA 11.2. *Let $G \ni (0,0)$ a finite shape of $\mathbb{N}$.*

If for all $k > 0$, there is an x_k such that there is only one y with $(x_k, y) \in (G^k)^+$ and $(x_k, y+1) \in (G^k)^+$, then G^∞ cannot be strictly self-assembled in the aTAM model unless $G = \{\vec{0}\}$.

Proof. Let w and h be the width and height of G. Assume that there is an aTAM $\mathcal{G}$ which self-assembles G^∞ at temperature τ.

Let C_k be the set of all glues which appear on the eastern edge of position (x_k, y) for some value of y. Since $\mathcal{G}$ assembles G^∞, all the C_k are non-empty, so let $C_\infty = \bigcap_k (\bigcup_{k'>k} C_{k'})$ be the set of glues appearing in infinitely many C_k; C_∞ is also non-empty. Let $F_k \subseteq C_k$ be the set of glues g such that there is a position $\vec{z} = (x_k, y)$ and a production $\Pi_{k,t}$ of $\mathcal{G}$ where:

- the eastern glue of $\Pi_{k,t}(\vec{z})$ is g
- $\Pi_{k,t}$ contains no tile right of x_k

Like C_∞, $F_\infty = \bigcap_k (\bigcup_{k'>k} F_{k'})$ is non-empty. The sets C_k and F_k are illustrated on Figure 43.

Now let t be a glue of $\mathcal{G}$, and s_t be a (new) tile with t on its eastern side, and ϵ on all other sides. Let $\mathcal{G}_t[x \geq 1]$ be the set of productions of $\mathcal{G}$, starting from the seed configuration with s_t at $\vec{0}$ and attaching no tiles at positions $x < 1$. Let $u_t = \min_{p \in \mathcal{G}_t[x>1]} \max\{y | (x,y) \in p\}$ and $d_t = \min_{p \in \mathcal{G}_t[x>1]} \min\{y | (x,y) \in p\}$. These two quantities are illustrated on Figure 44.

Since $G^\infty \subset \mathbb{N}^2$, it does not contain any infinite southward path. Therefore, for any k and $t \in F_k$, $d_t > -\infty$, otherwise from $\Pi_{k,t}$ it would be possible to produce paths going arbitrarily far down, out of $\mathbb{N}^2$. Let $d_\infty = \max_{t \in F_\infty} \{d_t | t \in \mathbb{G}\}$, d_∞ is finite. On the other hand, $u_\infty = \max\{u_t | t \in F_\infty\}$ must be infinite. Indeed, if it is finite, let $s = u_\infty - d_\infty + 1$. Consider a maximal production Π^* obtained by only placing tiles left of the vertical line L^* at x-coordinate x_s. Starting from Π^*, the only attachable positions are isolated points on L^*, separated by a distance at least w^s. From each of them, it is possible to grow an arm which reaches no higher than s upwards, whence, as illustruted on Figure 45, the part right of L^* of that production is contained within a union of $(u_\infty - d_\infty)$-width horizontal bands, so its domain cannot be G^∞.

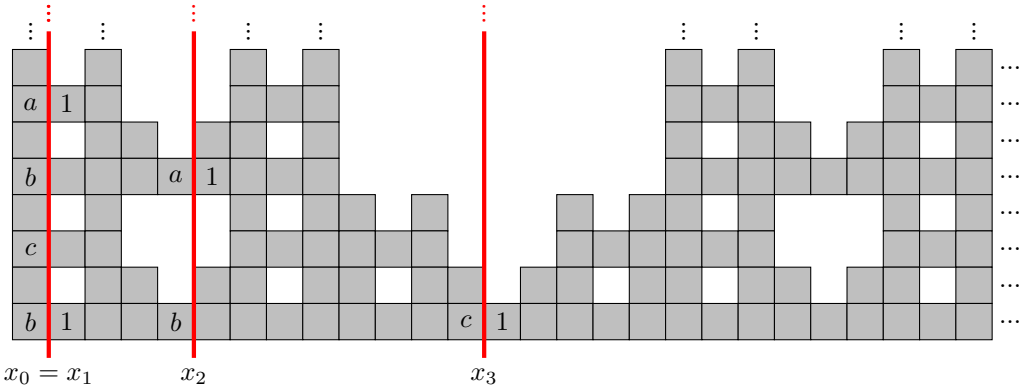


Figure 43: The set C_k is the set of all glues which attach across the vertical lines at coordinate x_k. Ignoring any tiles which are not visible in the picture, $C_0 = C_1 = \{a \cdot E, b \cdot E, c \cdot E\}$, $C_2 = \{a \cdot E, b \cdot E\}$ and $C_3 = \{c \cdot E\}$. The tiles marked with a 1 are the ones which appear in their column in some production without other tiles right of the red line; the subset F_k of C_k contains their east glues: , $F_0 = F_1 = \{a \cdot E, b \cdot E\}$, $C_2 = \{a \cdot E\}$ and $C_3 = \{c \cdot E\}$

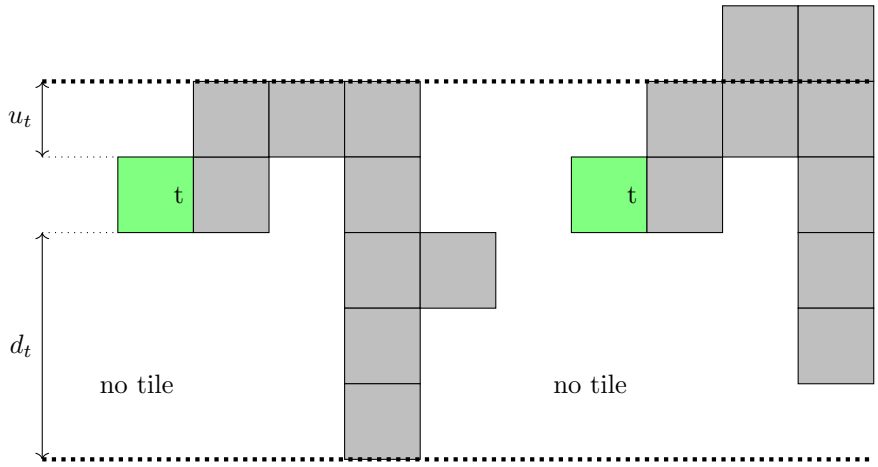


Figure 44: The definition of u_t and d_t given the two productions reachable from the glue t, without going left of the seed.

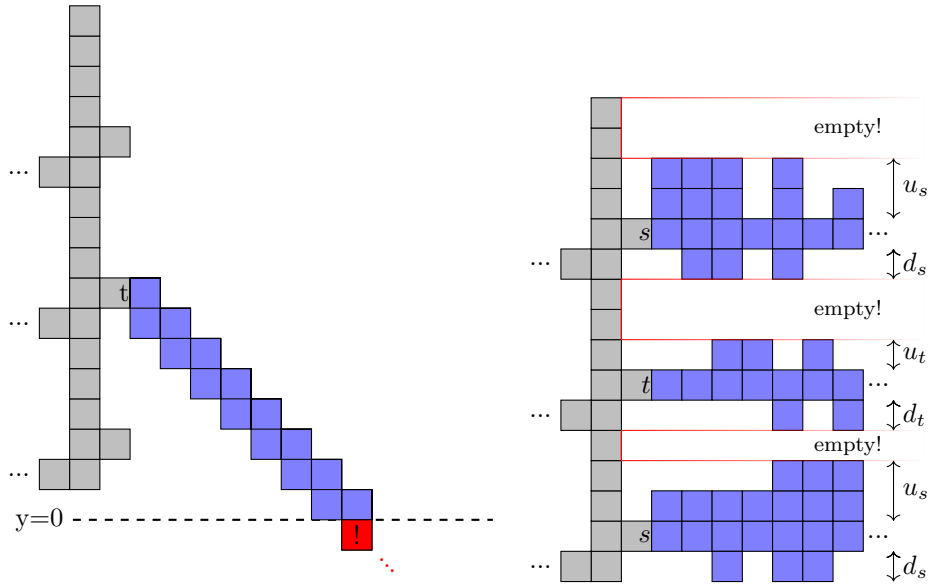


Figure 45: Faulty terminal productions which can be reached, left if some d_t is infinite, right if all u_t are finite: $\min_t(d_t) = -1$, $u_t = 1$, $u_s = 2$.

For t such that $u_t = +\infty$, all terminal productions on the right half-plane reach infinitely high. Thus, by applying theorem 3.4 with increasing values of m, either one of these terminal productions contains a periodic path going up, or there are productions P with arbitrarily large squares in their fill-in $P^\bullet$. In both cases, for each k, there is a production F_k for which some $k \times k$ square is inaccessible from any point $(0, y)$ with $y > 0$ without crossing F_k, as illustrated on Figure 46.

For a large enough level of substitution, any glue t for which u_t is finite does not even grow one "meta-cell" up. Let $F_! = \{g \in F_\infty | u_g = +\infty\}$, and $s_!$ be such that $\max\{u_g | g \notin F_!\} < h^{s_!}$ and $|d_\infty| < h^{s_!}$.

Consider a vertical line L_x at some position x between copies of $G^{s_!}$ in G^∞. Let P_y be a maximal subproduction of P which can be assembled without attaching any tile right of L_y. In P_y, let $\vec{z}_x = (x, y)$ be the lowest position on L_x containing a tile T of $F_{s!}$.

By the pigeonhole principle, there must be two vertical lines L_n and L_m between copies of $G^{s_!}$ such that $P_n(\vec{z}_n) = P_m(\vec{z}_m)$. Let $\vec{v}$ be the vector $\vec{z}_m - \vec{z}_n$. By construction of P_n and P_m, for any production $P_n^!$ obtained from P_n, there is a production $P_m^!$ obtained from P_m such that the part of $P_n^!$ above and to the right of $\vec{z}_n$ and the part of $P_m^!$ above and to the right of $\vec{z}_m$ are identically filled. Hence, there is a vector which preserves either arbitrarily large cycles of G^∞ or its intersection with a quarter-plane, thus G must be trivial. ∎

Still open is the case where in all the iterates $(G^k)^+$, there is a vertex which disconnects either the north and south or the east and west, but no straight line which crosses $(G^k)^+$ only once. A more precise variant of the previous proof takes care of that case.

THEOREM 11.1. *Let $G \ni (0,0)$ a finite shape of $\mathbb{N}$, and for $k \in \mathbb{N}$, $G^k = \sigma_G^k(\{(0,0)\})$.*

If $\sup_k(G^k[N \leftrightarrow S]) = 1$ or $\sup_k(G^k[E \leftrightarrow W]) = 1$, then G_∞ cannot be strictly self-assembled in the aTAM model unless $G^\infty = \mathbb{N}^2$.

Proof. Assume without loss of generality that $\sup(G^k[E \leftrightarrow W]) = 1$.

The proof is the same as that of Lemma 11.2, except that instead of straight lines cutting the fractal only once every h^k, one needs to consider cuts which pass through the first occurence of each level of the east-west bridges. These cuts are not necessarily straight, but each of them remains within a bounded vertical band, which suffices for the remainder of the proof. ∎

11.3 Deciding Bounded Bandwidth theorem 3.8 gives a dichotomy between generators for which the associated fractal can be strictly self-assembled and those for which it cannot. There is a polynomial time

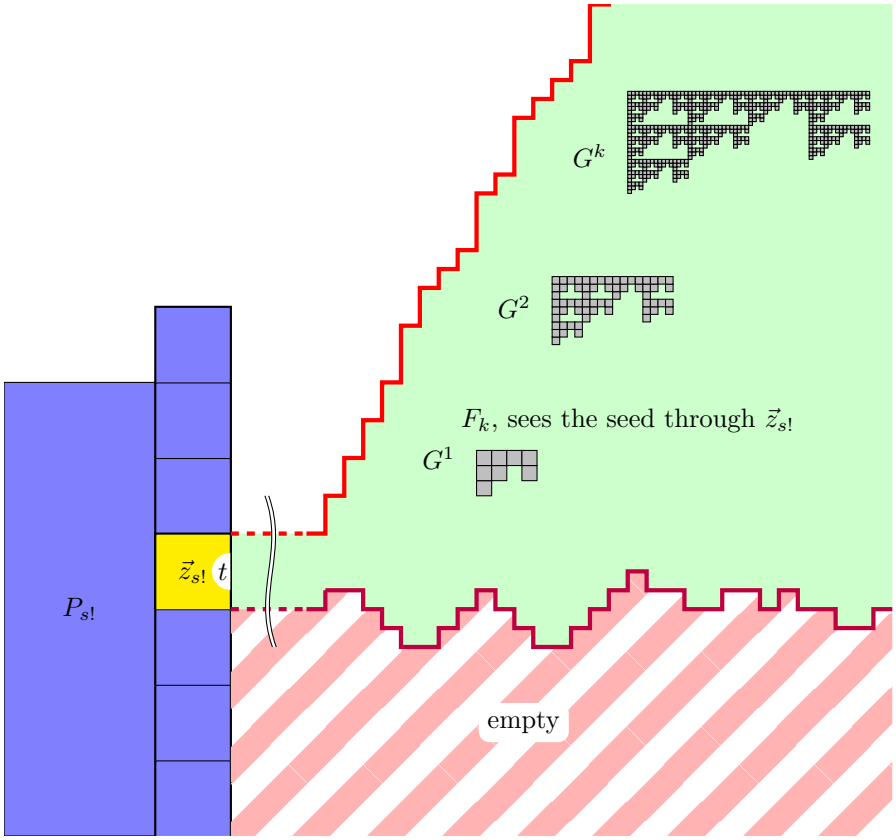


Figure 46: The part of the production that sees the seed only through $\vec{z}_!$ contains cycles of G^∞ of arbitrary size.

algorithm for determining which of Lemma 3.1 or theorem 3.8 applies.

THEOREM 11.2. *There is a polynomial time algorithm which, on input G decides whether:*

- *there is a k such that $G^k[N \leftrightarrow S] \geq 2$ and $G^k[E \leftrightarrow W] \geq 2$ and thus G^∞ can be strictly self-assembled in the aTAM, or*
- *for all k, $G^k[N \leftrightarrow S] < 2$ and thus G^∞ cannot be strictly self-assembled in the aTAM, or*
- *for all k, $G^k[E \leftrightarrow W] < 2$ and thus G^∞ cannot be strictly self-assembled in the aTAM.*

Notice that the second and third case are not *mutually exclusive.*

Proof. Without loss of generality, assume G is connected.

First, use a max-flow algorithm to compute $G[d \leftrightarrow d']$ for all directions d, d'. If both $G[E \leftrightarrow W]$ and $G[N \leftrightarrow S]$ are 2 or more, then the first case applies.

Let D be a set of pairs of directions, v is a D-disconnector if v disconnects G^{+d} from $G^{+d'}$ for any $(d, d') \in D$. If $G[E \leftrightarrow W] = 1$, then there is at least one $\{EW\}$-disconnector. For any D-disconnector, define $\text{Cause}(v, D)$ as the set of pair of directions (δ, δ') such that there are $(d, d') \in D$ and a path from G^{+d} to $G^{+d'}$ which enters v from direction δ and leaves it through direction δ'.

Compute the disconnect causation graph Δ. It is a directed graph whose vertices are couples (v, D) where v is a D-disconnector, and there is an arc from (v, D) to each vertex $(v', \text{Cause}(v, D))$. The causation graph Δ has size at most $64|G|$ and the existence of each of its arcs can be tested in polynomial time.

The graph Δ contains a cycle reachable from $(v, \{EW\})$, if and only if for all k, $G^k[E \leftrightarrow W] < 2$, likewise for N and S. Indeed, for $k > 0$, a vertex v is a D-disconnector in G^k if and only if:

- The vertex of $\lfloor v/G^{k-1} \rfloor$ of G corresponding to the copy of G^{k-1} containing v is a D-disconnector, and
- the vertex $(v \bmod G^{k-1})$ of G^{k-1} corresponding to the position of v within that copy is a $\text{Cause}(v, D)$-disconnector.

Hence, from Δ, it is possible to distinguish between the three cases. ∎

12 Open Questions

The tools we have introduced here —standard systems, quines, self-describing circuits— as well as our characterization of the behavior of bounded-treewidth productions in the aTAM through theorem 3.4 have passed the test of strict self-assembly of fractals, which has been an open question in self-assembly for more than a decade.

We are curious to know what the landscape of fractal self-assembly is in 3D. In cellular automata, Tilings, as well as in the aTAM, it is well-known that uncomputability can appear with the third dimension. While the Tree-Pump Theorem certainly would work in 3D, as well as on any "reasonable grid", in $\mathbb{Z}^3$, the periodic path obtained from it does not isolate a large domain. Hence, the question of characterizing DSSFs which can be strictly self-assembled in $\mathbb{Z}^3$ remains open.

The Tree Pump Theorem links *bounded* connected treewidth with having a periodic, and thus compuatble behavior. This leads to the question of quantitative bounds: can an analogue of Complexity Theory be built on treewidth for the aTAM? This also begs the question whether using connected treewidth is fundamental to this pumping principle: are there bounded treewidth, aperiodic shapes which can be strictly self-assembled?

References

[1] Andrew Alseth and Matthew J Patitz. "The need for seed (in the abstract tile assembly model)". In: *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2023, pp. 4540–4589.

[2] Robert D. Barish et al. "An information-bearing seed for nucleating algorithmic self-assembly". In: *Proceedings of the National Academy of Sciences* 106.15 (Apr. 2009), pp. 6054–6059. DOI: 10.1073/pnas.0808736106. URL: http://dx.doi.org/10.1073/pnas.0808736106.

[3] Kimberly Barth et al. "Scaled tree fractals do not strictly self-assemble". In: *Unconventional Computation & Natural Computation (UCNC) 2014, University of Western Ontario, London, Ontario, Canada* July 14-18, 2014. 2014, pp. 27–39.

[4] Florent Becker, Eric Rémila, and Nicolas Schabanel. "Time Optimal Self-assembly for 2D and 3D Shapes: The Case of Squares and Cubes". In: *DNA*. Ed. by Ashish Goel, Friedrich C. Simmel, and Petr Sosík. Vol. 5347. Lecture Notes in Computer Science. Springer, 2008, pp. 144–155. ISBN: 978-3-642-03075-8.

[5] Sarah Cannon et al. "On the effects of hierarchical self-assembly for reducing program-size complexity". In: *Theor. Comput. Sci.* 894 (2021), pp. 50–78.

[6] Sarah Cannon et al. "Two Hands Are Better Than One (up to constant factors): Self-Assembly In The 2HAM vs. aTAM." In: *STACS*. Ed. by Natacha Portier and Thomas Wilke. Vol. 20. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013, pp. 172–184. ISBN: 978-3-939897-50-7.

[7] Karel Culik and Jarkko Kari. "On aperiodic sets of Wang tiles". In: *Foundations of Computer Science: Potential — Theory — Cognition*. Ed. by Christian Freksa, Matthias Jantzen, and Rüdiger Valk. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 153–162. ISBN: 978-3-540-69640-7. DOI: 10.1007/BFb0052084. URL: https://doi.org/10.1007/BFb0052084.

[8] Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, Aug. 2005. ISBN: 3540261826. URL: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20%7B%5C&%7Dpath=ASIN/3540261826.

[9] Reinhard Diestel and Malte Müller. "Connected Tree-Width". In: *Comb.* 38.2 (2018), pp. 381–398. DOI: 10.1007/S00493-016-3516-5. URL: https://doi.org/10.1007/s00493-016-3516-5.

[10] David Doty et al. "The tile assembly model is intrinsically universal". In: *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science*. FOCS 2012. New Brunswick, New Jersey, 2012, pp. 302–310.

[11] David Doty et al. "Zeta-Dimension". In: *Proceedings of the Thirtieth International Symposium on Mathematical Foundations of Computer Science*. Springer-Verlag, 2005, pp. 283–294.

[12] Jérôme Durand-Lose et al. "Self-assembly of 3-D structures using 2-D folding tiles". In: *Nat. Comput.* 19.2 (2020), pp. 337–355. DOI: 10.1007/S11047-019-09751-9. URL: https://doi.org/10.1007/s11047-019-09751-9.

[13] Constantine Glen Evans. "Crystals that count! Physical principles and experimental investigations of DNA tile self-assembly". PhD thesis. California Institute of Technology, 2014.

[14] David Furcy, Scott M Summers, and Christian Wendlandt. "Self-assembly of and optimal encoding within thin rectangles at temperature-1 in 3D". In: *Theoretical Computer Science* 872 (2021), pp. 55–78.

[15] David Furcy and Scott M. Summers. "Scaled pier fractals do not strictly self-assemble". In: *Natural Computing* 16.2 (2017), pp. 317–338.

[16] Chaim Goodman-Strauss. "Matching rules and substitution tilings". In: *Annals of Mathematics* (1998), pp. 181–223.

[17] Howard Gutowitz. *Cellular automata: Theory and experiment*. MIT press, 1991.

[18] Daniel Hader. *WebTAS: A Browser-based Simulator*. 2024. URL: http://self-assembly.net/wiki/index.php/WebTAS.

[19] Daniel Hader, Matthew J Patitz, and Scott M Summers. "Fractal dimension of assemblies in the abstract tile assembly model". In: *Natural Computing* (2023), pp. 1–16.

[20] Daniel Hader and Matthew J. Patitz. *Strict self-assembly of discrete self-similar fractals*. 2024. URL: http://self-assembly.net/wiki/index.php/Strict_self-assembly_of_discrete_self-similar_fractals.

[21] Daniel Hader and Matthew J. Patitz. "The Impacts of Dimensionality, Diffusion, and Directedness on Intrinsic Cross-Model Simulation in Tile-Based Self-Assembly". In: *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*. Ed. by Kousha Etessami, Uriel Feige, and Gabriele Puppis. Vol. 261. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 71:1–71:19. DOI: 10.4230/LIPIcs.ICALP.2023.71. URL: https://doi.org/10.4230/LIPIcs.ICALP.2023.71.

[22] Daniel Hader et al. "The Impacts of Dimensionality, Diffusion, and Directedness on Intrinsic Universality in the abstract Tile Assembly Model". In: *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*. Ed. by Shuchi Chawla. SIAM, 2020, pp. 2607–2624.

[23] Jacob Hendricks, Matthew J. Patitz, and Trent A. Rogers. "Universal Simulation of Directed Systems in the abstract Tile Assembly Model Requires Undirectedness". In: *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2016), New Brunswick, New Jersey, USA* October 9-11, 2016. 2016, pp. 800–809.

[24] Jacob Hendricks et al. "Hierarchical Growth is Necessary and (Sometimes) Sufficient to Self-Assemble Discrete Self-Similar Fractals". In: *Proceedings of the 24th International Conference on DNA Computing and Molecular Programming (DNA 24), Shandong Normal University, Jinan, China* October 8-12. 2018, pp. 87–104.

[25] Jacob Hendricks et al. "Hierarchical growth is necessary and (sometimes) sufficient to self-assemble discrete self-similar fractals". In: *Natural Computing* 19.2 (June 1, 2020), pp. 357–374. ISSN: 1572-9796. DOI: 10.1007/s11047-019-09777-z. URL: https://doi.org/10.1007/s11047-019-09777-z (visited on 10/24/2023).

[26] James I. Lathrop, Jack H. Lutz, and Scott M. Summers. "Strict Self-Assembly of Discrete Sierpinski Triangles". In: *Theoretical Computer Science* 410 (2009), pp. 384–405.

[27] James I. Lathrop et al. "Computability and Complexity in Self-assembly". In: *Theory Comput. Syst.* 48.3 (2011), pp. 617–647.

[28] Jack H. Lutz and Brad Shutters. "Approximate Self-Assembly of the Sierpinski Triangle". In: *Theory Comput. Syst.* 51.3 (2012), pp. 372–400.

[29] Pierre-Etienne Meunier et al. "Intrinsic universality in tile self-assembly requires cooperation". In: *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SODA '14. USA: Society for Industrial and Applied Mathematics, Jan. 5, 2014, pp. 752–771. ISBN: 978-1-61197-338-9. (Visited on 03/19/2024).

[30] Pierre-Étienne Meunier, Damien Regnault, and Damien Woods. "The program-size complexity of self-assembled paths". In: *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*. Ed. by Konstantin Makarychev et al. ACM, 2020, pp. 727–737. DOI: 10.1145/3357713.3384263. URL: https://doi.org/10.1145/3357713.3384263.

[31] Pierre-Étienne Meunier and Damien Woods. "The Non-cooperative Tile Assembly Model is Not Intrinsically Universal or Capable of Bounded Turing Machine Simulation". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2017. Montreal, Canada: ACM, 2017, pp. 328–341. ISBN: 978-1-4503-4528-6. DOI: 10.1145/3055399.3055446. URL: http://doi.acm.org/10.1145/3055399.3055446.

[32] Pierre-Étienne Meunier and Damien Woods. "The non-cooperative tile assembly model is not intrinsically universal or capable of bounded Turing machine simulation". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*. 2017, pp. 328–341. DOI: 10.1145/3055399.3055446. URL: http://doi.acm.org/10.1145/3055399.3055446.

[33] Melanie Mitchell et al. "Computation in Cellular Automata: A Selected Review." In: *Non-standard computation* (2005), pp. 95–140.

[34] Matthew J Patitz and Scott M Summers. "Self-assembly of infinite structures". In: *International Workshop on The Complexity of Simple Programs (CSP 2008), University College Cork, Ireland*. Ed. by Turlough Neary et al. Cork University Press, Dec. 2008, pp. 279–291.

[35] Matthew J Patitz and Scott M Summers. "Self-assembly of infinite structures: A survey". In: *Theoretical computer science* 412.1-2 (2011), pp. 159–165.

[36] Matthew J. Patitz and Scott M. Summers. "Self-assembly of decidable sets". In: *Natural Computing* 10.2 (2011), pp. 853–877.

[37] Matthew J. Patitz and Scott M. Summers. "Self-Assembly of Discrete Self-Similar Fractals". In: *Natural Computing* 1 (9 2010), pp. 135–172.

[38] Roger Penrose. "Pentaplexity a class of non-periodic tilings of the plane". In: *The mathematical intelligencer* 2 (1979), pp. 32–37.

[39] John H. Reif. "Molecular Assembly and Computation: From Theory to Experimental Demonstrations". In: *Proceedings of the Twenty-Ninth International Colloquium on Automata, Languages and Programming*. 2002, pp. 1–21.

[40] Paul W. K. Rothemund and Erik Winfree. "The Program-size Complexity of Self-Assembled Squares (extended abstract)". In: *STOC '00: Proceedings of the thirty-second annual ACM Symposium on Theory of Computing*. Portland, Oregon, United States: ACM, 2000, pp. 459–468.

[41] Paul WK Rothemund, Nick Papadakis, and Erik Winfree. "Algorithmic self-assembly of DNA Sierpinski triangles". In: *PLoS biology* 2.12 (2004), e424.

[42] Rebecca Schulman, Christina Wright, and Erik Winfree. "Increasing Redundancy Exponentially Reduces Error Rates during Algorithmic Self-Assembly". In: *ACS Nano* 9.6 (2015). PMID: 25965580, pp. 5760–5771. DOI: 10.1021/nn507493s.

[43] Rebecca Schulman, Bernard Yurke, and Erik Winfree. "Robust self-replication of combinatorial information via crystal growth and scission". In: *Proceedings of the National Academy of Sciences* 109.17 (2012), pp. 6405–10. ISSN: 1091-6490. URL: http://www.biomedsearch.com/nih/Robust-self-replication-combinatorial-information/22493232.html.

[44] Nadrian C. Seeman et al. "Logical computation using algorithmic self-assembly of DNA triple-crossover molecules". In: *Nature* 407.6803 (Sept. 2000), pp. 493–496. DOI: 10.1038/35035038.

[45] David Soloveichik and Erik Winfree. "Complexity of Self-Assembled Shapes". In: *SIAM Journal on Computing* 36.6 (2007), pp. 1544–1569.

[46] William P. Thurston. "Conway's Tiling Groups". In: *The American Mathematical Monthly* 97.8 (1990), pp. 757–773. DOI: 10.1080/00029890.1990.11995660. URL: https://doi.org/10.1080/00029890.1990.11995660.

[47] Erik Winfree. "Algorithmic Self-Assembly of DNA". PhD thesis. California Institute of Technology, June 1998.

[48] Damien Woods. "Intrinsic universality and the computational power of self-assembly". In: *MCU: Proceedings of Machines, Computations and Universality*. Vol. 128. Univ. of Zürich, Switzerland. Sept. 9-12: Open Publishing Association, 2013, pp. 16–22.

[49] Damien Woods. "Intrinsic universality and the computational power of self-assembly". In: *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 373.2046 (2015). ISSN: 1364-503X. DOI: 10.1098/rsta.2014.0214. eprint: http://rsta.royalsocietypublishing.org/content/373/2046/20140214.full.pdf. URL: http://rsta.royalsocietypublishing.org/content/373/2046/20140214.

[50] Damien Woods et al. "Diverse and robust molecular algorithms using reprogrammable DNA self-assembly". In: *Nature* 567.7748 (2019), pp. 366–372.