Universal Shape Replication via Self-Assembly with Signal-Passing Tiles

Andrew Alseth
University of Arkansas, Fayetteville, AR, USA

Daniel Hader
University of Arkansas, Fayetteville, AR, USA

Matthew J. Patitz
University of Arkansas, Fayetteville, AR, USA

Abstract

In this paper, we investigate shape-assembling power of a tile-based model of self-assembly called the Signal-Passing Tile Assembly Model (STAM). In this model, the glues that bind tiles together can be turned on and off by the binding actions of other glues via "signals". In fact, we prove our positive results in a version of the model in which it is slightly more difficult to work (where tiles are allowed to rotate) but show that they also hold in the standard STAM. Specifically, the problem we investigate is "shape replication" wherein, given a set of input assemblies of arbitrary shape, a system must construct an arbitrary number of assemblies with the same shapes and, with the exception of size-bounded junk assemblies that result from the process, no others. We provide the first fully universal shape replication result, namely a single tile set capable of performing shape replication on arbitrary sets of any 3-dimensional shapes without requiring any scaling or pre-encoded information in the input assemblies. Our result requires the input assemblies to be composed of signal-passing tiles whose glues can be deactivated to allow deconstruction of those assemblies, which we also prove is necessary by showing that there are shapes whose geometry cannot be replicated without deconstruction. Additionally, we modularize our construction to create systems capable of creating binary encodings of arbitrary shapes, and building arbitrary shapes from their encodings. Because the STAM is capable of universal computation, this then allows for arbitrary programs to be run within an STAM system, using the shape encodings as input, so that any computable transformation can be performed on the shapes.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Algorithmic self-assembly, Tile Assembly Model, shape replication

Digital Object Identifier 10.4230/LIPIcs.DNA.2022.2

Category Extended Abstract

Related Version *Full Version*: `https://arxiv.org/abs/2206.03908`

Funding This work was supported in part by NSF grant CAREER-1553166.

1 Introduction

Artificial self-assembling systems are most often designed with the goal of building structures "from scratch". That is, they are designed so that they will start from a disorganized set of relatively simple components (often abstractly called *tiles*) that autonomously combine to form more complex target structures. This process often begins from collections of only unbound, singleton tiles, or sometimes also includes so-called *seed assemblies* which may be small (in relation to the target structure) "pre-built" assemblies that encode some information which *seeds* the growth of larger assemblies. This growth occurs as additional tiles bind to those seed assemblies according to the rules of the system, allowing them to eventually grow into the desired structures. Examples have been shown in both experimental

settings (e.g. [14, 37, 22]), as well as in the mathematical domains of abstract models (e.g. [34, 32, 7, 12, 10]). However, in the subdomain of algorithmic self-assembly, in which systems are designed so that the tile additions implicitly follow the steps of pre-designed algorithms, other goals have also been pursued. These have included, for instance, performing computations (e.g. [24, 30]), identifying input assemblies that match target shapes [31], replicating patterns on input assemblies [23, 33], and replicating (the shapes of) input assemblies [6, 26, 1, 3, 17]. In this paper, we explore the latter, particularly the theoretical limits of systems within a mathematical model of self-assembling tiles to replicate shapes.

We use the term *shape replication* to refer to the goal of designing self-assembling systems that take as input seed assemblies and which produce new assemblies that have the same shapes as those seed assemblies [1]. In order for tile-based self-assembling systems to perform shape replication, dynamics beyond those of the original abstract Tile Assembly Model (aTAM), introduced by Winfree [36] and widely studied (e.g. [34, 32, 12, 24, 5, 27, 18, 25]), are required. In the aTAM, tiles attach to the seed assembly and the assemblies which grow from it, one tile at a tile, and tile attachments are irreversible. A generalization of the aTAM, the hierarchical assembly model known as the 2-Handed Assembly Model [5, 7], allows for the combination of pairs of arbitrarily large assemblies, but it too only allows irreversible attachments. However, for shape replication, it is fundamentally important that at least some tiles are able to bind to the input assemblies to gather information about their shapes which is then used to direct the formation of the output assemblies, since binding to an assembly is the only mechanism for interacting with it. These output assemblies eventually must not be connected to the input assemblies if they are to have the same shapes as the original input assemblies. This requires that at some point tile bindings can be broken. A number of theoretical models have been proposed with mechanisms for breaking tiles apart, for example: glues with repulsive forces [29, 26], subsets of tiles which can be dissolved at given stages of assembly [1, 11], tiles which can turn glues on and off [28, 20] (a.k.a. *signal-passing tiles*), and systems where the temperature can be increased to cause bonds to break [7, 35]. Within these models, previous results have shown the power of algorithmic self-assembling systems to perform shape replication. In [6], they used glues with repulsive forces, and in [1] they used the ability to dissolve away certain types of tiles at given stages during the self-assembly process, and each showed how to replicate a large class on two-dimensional shapes. In [17], signal-passing tiles were shown to be capable of replicating arbitrary hole-free two-dimensional shapes if they are scaled up by a factor of 2. The results of [3] deal with the replication of three-dimensional shapes, and will be further discussed below.

The results of this paper are the first which provide for shape replication of all 3-dimensional shapes with no requirement for scaling those shapes. Additionally, although in [3] all three-dimensional shapes can be replicated at the small scale factor of 2, there it is necessary for the input assemblies to have relatively complex information embedded within them (in the form of Hamiltonian paths through all of their points being encoded by their glues). In our results, the input assemblies require no such embedded information. Furthermore, the model used in [3] is more complex, allowing not only for hierarchical assembly and signal-passing tiles, but also for tiles of differing shapes, and glue bindings that are flexible and thus allow for assemblies to reconfigure by folding. For the results of this paper, we have not only limited the dynamics to those of the Signal-Passing Tile Assembly Model (STAM), but have even placed an additional restriction on the model. Rather than assigning fixed orientations to tiles, in the model we use and call the STAM^R (i.e. the "STAM with rotation") tiles and assemblies are allowed to rotate. This allows us to consider an even more general, and difficult, version of the shape replication problem. Namely, the

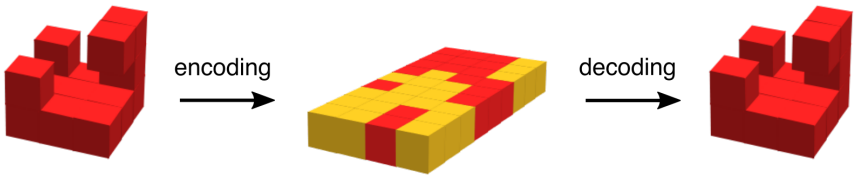


Figure 1 Schematic depiction of shape replication: (Left) An input assembly, (Middle) The assembly resulting from the encoding process which deconstructs the input assembly and encodes its shape, (Right) The assembly created by the decoding process, which uses the encoding as its input.

input assemblies in our constructions have glues of a single generic type covering their entire exteriors, and there is no distinction between a north-facing glue and an east-facing glue, for instance, as there is in the standard STAM. This makes several aspects of working with such generic input assemblies more difficult, but it is notable that our constructions need only trivial, simplifying modifications to work in the standard STAM and that our positive results thus also hold for the STAM. We show that there is a "universal shape replicator" which is a tileset in the STAM^R that can be used in conjunction with any set of generic input assemblies and will cause assemblies of every shape in the input set to be simultaneously produced in parallel. This is the first truly universal shape replicator for two or three dimensional shapes[1]. Furthermore, we break our construction into two major components, a "universal encoder" and a "universal decoder" (see Figure 1 for a depiction). The universal encoder is capable of taking generic input assemblies and creating assemblies that expose binary sequences that encode those shapes, and the universal decoder is capable of taking assemblies exposing those encodings and creating assemblies of the encoded shapes. Due to the Turing universality of this model, this also allows for the full range of all possible computational transformations to occur between the encoding and decoding, and thus enables the generation of any transformations of the shapes of the input assemblies, such as creating scaled versions or complementary shapes.

In order for our universal shape replication construction to operate, the input assemblies must be created from signal-passing tiles which are capable of turning off their glues and dissociating from the assemblies. This allows for the assemblies to be "deconstructed", and we prove that this is necessary in order to replicate arbitrary shapes, specifically those which have enclosed or narrow, curved cavities, and this is intuitively clear since otherwise there would be no way to determine which locations in the interior of an input shape are included in the shape, and which are part of an enclosed void. Our proof that it is also impossible to replicate shapes with curved, but not enclosed, cavities further exhibits the additional difficulty of working within the STAM^R model which allows tile rotations.

While our universal shape encoder, decoder, and replicator achieve the full goal of the line of research into shape replication, and also provide the ability to augment shape-building with arbitrary computational transformations, we note that the results are highly theoretical and serve more generally as an exploration of the theoretical limits of self-assembling systems. The tilesets are relatively large and require tiles with large numbers of signals, and although the input assemblies are not required to have complex information embedded within them, a trade-off that occurs compared with the results of [3] is that our constructions make use

[1] Note that while replicating two-dimensional shapes, which consist of points in a single plane, our construction will utilize three dimensions.

of a large amount of "fuel". That is, a large number of tiles are used during various phases but they are only temporary and aren't contained within the target assemblies and thus are "consumed" by the construction process. Despite the complexity of these theoretical constructions, we think that several modules and techniques developed may be of future use within other constructions (e.g. our "leader election" procedure which is guaranteed to uniquely select a single corner of an input assembly's bounding prism, to serve as a staring location for our encoding procedure within a constant number of assembly steps despite the lack of directional information provided by such an assembly), and also that these results may lead the way to similarly powerful but less complex constructions that may eventually achieve a level of being physically plausible to construct.

This paper is organized as follows. In Section 2 we provide an overview of the STAM^R and definitions. In Section 3 we state our main theorem and supporting lemmas, and discuss the constructions that prove them. In Section 4 we briefly describe some of the computational transformations that could be used to augment our constructions, and in Section 5 we prove deconstruction is necessary for shape replication of certain classes of shapes. Due to space constraints, full details can be found in [4] and the Appendix contains more details of the STAM^R and a series of subconstructions that appear throughout the constructions.

2 Definitions

In this section we provide definitions of the model used, and also for several of the terms used throughout the paper.

2.1 Overview of the STAM^R model

Here we provide only a high-level overview of the STAM^R model since it is so similar to the standard STAM model [28]. A full definition can be found in Section A.1 of the Appendix. The STAM^R is based on the STAM, which is based on the 2-Handed Assembly Model (a.k.a. Hierarchical Assembly Model), in which the fundamental components are *tiles* which have *glues* on their sides that allow them to bind together. Here, tiles are 3-dimensional unit cubes. Each glue has a string *label* and an integer *strength*, and glues can bind to each other when they are adjacent and have the same strength and have complementary labels (which we denote using "$*$", e.g. "l_1" and "l_1^*"). There is a system parameter $\tau \in \mathbb{Z}^+$ which determines the strength threshold that must be reached for tiles to bind together. Individual tiles may bind, and also any pair of *assemblies* (groups of previously bound tiles, also called *supertiles*) may bind if they can bind with at least τ strength summed across bonds and don't have overlapping tiles. Tiles and assemblies are free to rotate and don't have fixed orientations. The glues of tiles may be assigned *signals* which *fire* when those glues form bonds. These signals can turn glues on the same tile either `on` or `off`. The time between which a signal is fired and the glue it targets changes its state is nondeterministic. Each signal can fire only one time, no matter how many times its glue may form bonds. Turning glues `off` may cause previously τ-stable assemblies (ie. those where all subassemblies are connected by $\geq \tau$ strength) to break apart. When our constructions cause tiles to turn `off` glues and dissociate from assemblies, we often use the term *dissolve* since the tile is "removed" from the assembly, although it is not technically dissolved. A *system* in the STAM^R is a triple (T, S, τ) where T is a set of tile types, S is an initial state consisting of a multiset of tiles and *input* (or *seed*) assemblies and their counts (which may be infinite). τ is the binding threshold as previously discussed. An *assembly sequence* is a (possibly infinite) series of discrete steps beginning fron the initial state of a system that may include (super)tile bindings, changes in glue states, or

Figure 2 Example of a bent cavity, assuming that the planes on the sides into and out of the page were also filled in, leaving a single-cube-wide path into the interior of the shape.

breaking of assemblies. The *producible assemblies* of a system $\mathcal{T}$, denoted $\mathcal{A}[\mathcal{T}]$, are those which can result from the steps of some assembly sequence. The *terminal assemblies*, denoted $\mathcal{A}_{\square}[\mathcal{T}]$ are those which can be produced but for which no other valid actions may occur.

▶ **Definition 1.** *Given an STAM^R system $\mathcal{T} = (T, S, \tau)$, we say that it* finitely completes *with respect to a set of terminal assemblies $\hat{\alpha}$ if and only if there exists some constant $c \in \mathbb{N}$ such that, if in the initial configuration S, each element of S was assigned count c, in every possible valid assembly sequence of $\mathcal{T}$, every element of $\hat{\alpha}$ is produced.*

A system which finitely completes with respect to assemblies $\hat{\alpha}$ is guaranteed to always produce those assemblies as long as it begins with enough copies of the (super)tiles in its initial configuration, i.e. it cannot follow any assembly sequence which would consume one or more (super)tiles needed to form those assemblies before making them.

▶ **Definition 2.** *A* shape *is a non-empty connected subset of $\mathbb{Z}^3$, i.e. a connected set of unit cubes each of which is centered at a coordinate $\vec{v} \in \mathbb{Z}^3$. A* finite shape *is a finite connected subset of $\mathbb{Z}^3$.*

In this paper, we consider shapes to be equivalent up to rotation and translation and unless stated otherwise explicitly, we will use the word *shape* to refer only to *finite shapes.*

▶ **Definition 3.** *Given a shape s, a* bounding box *is a rectangular prism in $\mathbb{Z}^3$ which completely contains s. The* minimum bounding box *is the smallest such rectangular prism.*

▶ **Definition 4.** *Given a shape s, we use the term* enclosed cavity *in s to refer to a set of connected points in $\mathbb{Z}^3$ that are not contained in s and for which no path in $\mathbb{Z}^3$ exists that does not intersect at least one point in s and gets infinitely far from all points in s.*

▶ **Definition 5.** *Given a shape s, we use the term* bent cavity *in s to refer to a set of connected points in $\mathbb{Z}^3$ contained inside of the minimum bounding box of s, b_s, but not contained within s itself, such that it includes some points which can be reached by straight lines in $\mathbb{Z}^3$ beginning from points in b_s, and some points which cannot be reached by straight lines in $\mathbb{Z}^3$ beginning from points in b_s.*

See Figure 2 for an example of a bent cavity.

▶ **Definition 6.** *We define a* shape encoding function *f_e as a function which, given as input an arbitrary shape s, returns a unique finite set E of binary strings, each unique for the shape s, such that there exists a* shape decoding function*, f_d and $f_d(e) = s$ for all $e \in E$.*

The shape encoding function we will define by construction in the proof of Lemma 14 will generate a set of binary strings for each input shape s such that each string encodes the points of the shape starting from a different reference corner and rotation of a bounding box. That can lead to up to 24 unique binary strings (for 3 rotations of each of 8 corners) for most shapes, but less for those with symmetry.

▶ **Definition 7.** *Given a shape S and a point $p = (x, y, z) \in S$, we define the* neighborhood *of p in S to be the set $S \cap \{(x+1, y, z), (x-1, y, z), (x, y+1, z), (x, y-1, z), (x, y, z+1), (x, y, z-1)\}$. We also say that neighborhoods are equivalent up to rotation, so there is 1 neighborhood containing 1 point, 2 with 2 points, 2 with 3 points, 2 with 4 points, 1 with 5 points, and 1 with 6 points.*

▶ **Definition 8.** *We define a* uniformly covered assembly *as an assembly α where every exposed side of every tile has the same strength 1 glue which is* `on`*. Additionally, if s is the shape of α, we require that for every 2 points $p, q \in s$ with the same neighborhood, a tile of the same type is located in both locations p and q in α.*

A uniformly covered assembly has the same glue all over its surface, with no glues marking special or unique locations, and has the same tile type in each location with the same neighborhood, so such an assembly can convey no information specific to particular locations, orientation, etc.

▶ **Definition 9.** *We define a* deconstructable assembly *as an assembly where (1) all neighboring tiles are bound to each other by one or more glues whose strengths sum to $\geq \tau$, and (2) each tile contains the glue(s) and signal(s) necessary to allow for all glues binding it to its neighbors to be turned* `off`*.*

In the following definitions, we will use the term *junk assembly* to refer to an assembly that is not a "desired product" of a system, but which is a small assembly composed of tiles which were used to facilitate the construction but are now terminal and cannot interact any further.

▶ **Definition 10** (Universal shape encoder)**.** *Let $\mathcal{S}$ be the set of all finite shapes, let f_e be a shape encoding function, let $c \in \mathbb{N}$ be a constant, and let E be a tileset in the STAM^R. If, for every finite subset of shapes $S' \subset S$, there exists an STAM^R system $\mathcal{E}_{S'} = (E, \sigma_{S'}, \tau)$, where $\sigma_{S'}$ consists of infinite copies of assemblies of each shape $s \in S'$ and also infinite copies of the singleton tiles from E, such that (1) for every shape $s \in S'$ there exists at least one binary string $b_s \in f_e(s)$ and there exist infinite terminal assemblies of $\mathcal{E}_{S'}$ that contain glues in the* `on` *state on the exterior surfaces of those assemblies that encode b_s (which we refer to as an* assembly encoding *s), (2) every terminal assembly is either an assembly encoding some $s \in S'$ or a "junk assembly" whose size is bounded by c, and (3) no non-terminal assembly grows without bound, then we say that E is a* universal shape encoder *with respect to f_e.*

▶ **Definition 11** (Universal shape decoder)**.** *Let $\mathcal{S}$ be the set of all finite shapes, let f_e be a shape encoding function, let $c \in \mathbb{N}$ be a constant, and let D be a tileset in the STAM^R. If, for every finite subset of shapes $S' \subset S$, there exists an STAM^R system $\mathcal{D}_{S'} = (D, \sigma_{S'}, \tau)$, where $\sigma_{S'}$ consists of infinite copies of assemblies each of which encode a shape $s \in S'$ with respect to f_e, and also infinite copies of the singleton tiles from D, such that (1) for every shape $s \in S'$ there exist infinite terminal assemblies of shape s, (2) every terminal assembly is either an assembly of the shape of some $s \in S'$ or a "junk assembly" whose size is bounded by c, and (3) no non-terminal assembly grows without bound, then we say that D is a* universal shape decoder *with respect to f_e.*

▶ **Definition 12** (Universal shape replicator)**.** *Let $\mathcal{S}$ be the set of all finite shapes and let R be a tileset in the STAM^R, and let $c \in \mathbb{N}$ be a constant. If, for every finite subset of shapes $S' \subset S$, there exists an STAM^R system $\mathcal{R}_{S'} = (R, \sigma_{S'}, \tau)$, where $\sigma_{S'}$ consists of infinite copies of assemblies of each shape $s \in S'$ and also infinite copies of the singleton tiles from R, such that (1) for every shape $s \in S'$ there exist infinite terminal assemblies of shape s,*

(2) every terminal assembly is either an assembly of the shape of some $s \in S'$ or a "junk assembly" whose size is bounded by c, (3) the number of assemblies of each shape $s \in S'$ grows infinitely, and (4) no non-terminal assembly grows without bound, then we say that R is a universal shape replicator*.*

3 3D Shape Replication

In this section, we state our main result, namely that there is a tileset in the STAM^R which is capable of replicating arbitrary shapes. This is formally stated in Theorem 13, and our proof works by providing modular constructions capable of encoding and decoding arbitrary sets of shapes which are given by Lemma 14 and Lemma 15, respectively, and then discussing how they can be combined to replicate shapes.

▶ **Theorem 13.** *There exists a tileset R in the STAM^R which is a universal shape replicator, such that for the systems using R (1) all input assemblies are uniformly covered, (2) the constant c which bounds the size of the junk assemblies equals 4, and (3) they finitely complete with respect to a set of terminal assemblies with the same shapes as the input assemblies.*

▶ **Lemma 14.** *There exist a shape encoding function f_e, and a tileset E in the STAM^R which is a universal shape encoder with respect to f_e, such that for the systems using E (1) all input assemblies are uniformly covered, (2) the constant c which bounds the size of the junk assemblies equals 4, and (3) they finitely complete with respect to a set of terminal assemblies which encode the shapes of the input assemblies.*

▶ **Lemma 15.** *There exist a shape decoding function f_d, and a tileset D in the STAM^R which is a universal shape decoder with respect to f_d, such that for the systems using D (1) the constant c which bounds the size of the junk assemblies equals 3 and (2) they finitely complete with respect to a set of terminal assemblies with the same shapes as those encoded by the input assemblies.*

3.1 Universal shape encoding

Here we describe the process by which a set of arbitrary shapes $S = \{s_1, \ldots, s_n\}$ can be encoded in the STAM^R using a universal shape encoding tileset E. We define our STAM^R system $\mathcal{E}_S$ to be the triple $(E, \Sigma_S, \tau = 2)$ where Σ_S is our initial system state consisting of all tiles in E, each with an infinite count, and additionally consists of a set $A = \{\alpha_1, \ldots, \alpha_n\}$ of uniformly covered, deconstructable assemblies such that the shape of α_i is s_i for $i = 1, \ldots, n$. The assemblies of A are called our *shape assemblies* and are made only of tiles from a fixed subset of E called *shape tiles*. Note that the glues and signals defined in these shape tiles are not used to encode any information regarding the structure of our shape assemblies; any shape specific information is inferred during the encoding process and the shape tiles simply contain the necessary glues and signals to perform basic tasks required for the encoding process, none of which are specific to any particular part of the shape assemblies.

The encoding process described below can largely be broken down into 3 steps. First, a bounding box is constructed around the shape assemblies using special tiles which are distinct from the shape tiles. Then, one of the corners of the box is elected non-deterministically to be the *leader corner* to provide an origin point which will represent the first tile location of our encoding. Finally, from the leader corner, the shape will be disassembled tile-by-tile during which an encoding assembly will be constructed, recording for each disassembled tile whether it is part of the shape or not (i.e. a "filler" tile used to assist the construction).

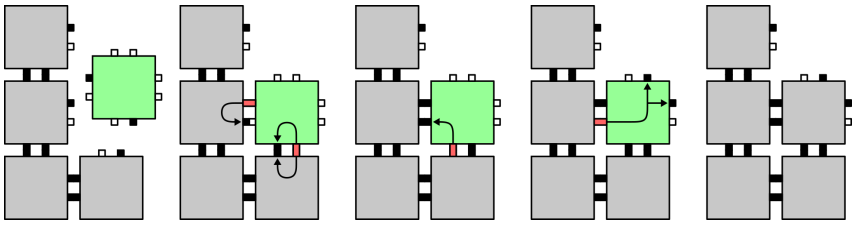


Figure 3 Filler tile binding to a concave site. Once a filler tile attaches cooperatively, signals activate glues on the filler tile and adjacent tiles. These glues ensure that the filler tile is attached with strength 2 on all sides. These glues are activated sequentially and once both are in the `on` state, signals activate output glues on all other sides of the filler tile. Once these signals activate, the supertile has 1 fewer concave site and the filler tile behaves as though it is just another tile on the supertile. While depicted in 2D for clarity, this occurs in 3D during our construction, but the idea is the same.

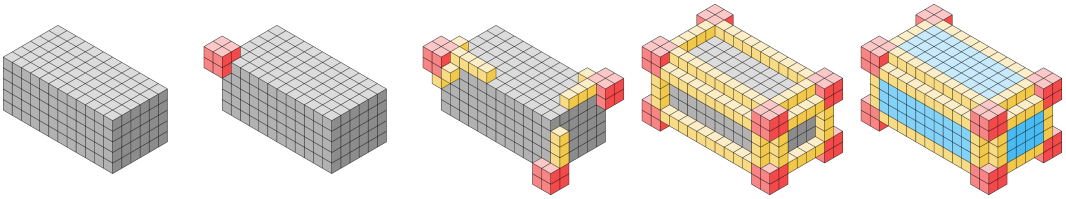


Figure 4 Growth of the inner shell around a bounding box (illustrated in gray). Growth begins by the attachment of corner gadgets (red). Cooperative binding with the corner gadgets and bounding box allow edge tiles to attach (yellow). Cooperation between the edge tiles and the bounding box then allows filler verification tiles (blue) to grow which are used to fill in the faces of the inner shell. The process by which these verification tiles bind to the bounding box ensures that there are no gaps or protrusions on the bounding box surface.

3.1.1 Bounding box assembly construction and verification

The first step in our encoding process begins by forming a bounding box assembly through the attachment of special tiles, called *filler tiles*, to a shape assembly. These filler tiles cooperatively bind to 2 diagonally adjacent tiles of our shape assembly in order to fill out any concave portions. (See Figure 3.) When a filler tile attaches to an assembly, signals are fired from the newly bound glues which activate additional glues between the filler tile and shape assembly. These new glues ensure that the filler tile is bound strongly on each face to the shape assembly. Glues are then activated on the other faces of the filler tile to further allow additional filler tiles to attach. Eventually, after sufficiently many filler tiles have attached, there will be no more locations in which another filler tile can attach. There are often many ways in which this can occur for any shape assembly, but the resulting bounding box assembly will always be a minimal bounding box of our shape. It should be noted that it's possible that not every location within the bounding box is filled. This can occur if the original shape had enclosed cavities, but can also occur because the attachment of filler tiles can create additional cavities as they attach. This is not a problem and it will always be possible for filler tiles to complete the outer surface of the bounding box.

In order to verify that the bounding box is fully formed, a shell of tiles is grown around our assembly. This shell, which we call the *inner shell*, is able to complete only if and only if the assembly is a fully formed bounding box. Figure 4 illustrates the high-level construction of the inner shell around a fully formed bounding box. Growth of the inner shell begins with the attachment of corner gadgets to our assembly. These corner gadgets then allow

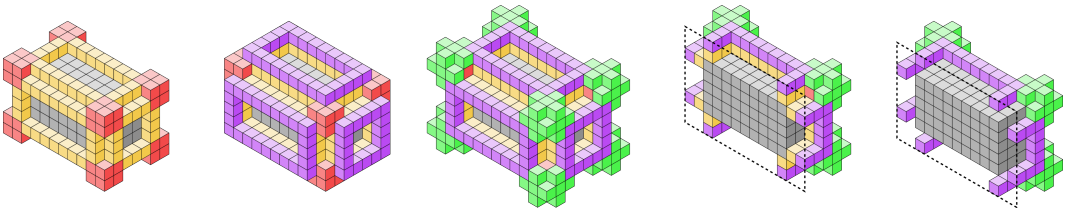


Figure 5 Once filler verification has successfully occurred on a surface of our bounding box, outer shell tiles attach to the edge tiles and corner gadgets on that surface to form a rectangle. Between the corners of these rectangles, outer corner gadgets can cooperatively bind. Once the corner gadgets have attached sufficiently to the outer shell tiles and the necessary connectivity conditions have been met, inner shell tiles are dissolved from between the outer shell and bounding box assembly. Illustrated using a cross-section view, the detachment of these tiles leaves us with a detached bounding box assembly that is too large to fit in the gaps of the outer shell, but too small to touch more than one interior corner of the outer shell simultaneously. Because of this, the bounding box assembly can then bind to an interior corner of the outer shell, but only on one corner, which is then elected leader.

for the cooperative attachment of edge tiles along the edges of the bounding box. Inside the region bounded by the edge tiles, special tiles called *filler verification tiles* cooperatively bind to the surface of the bounding box. If the surface of the bounding box assembly is not complete or contains gaps or protrusions to which additional filler tiles can bind, special gadgets called *detector gadgets* are able to bind to the erroneous edge tile or verification tile. This attachment causes a signal to fire which propagates along the erroneous tiles and causes them to deactivate their glues and detach.

3.1.2 Outer Shell Construction

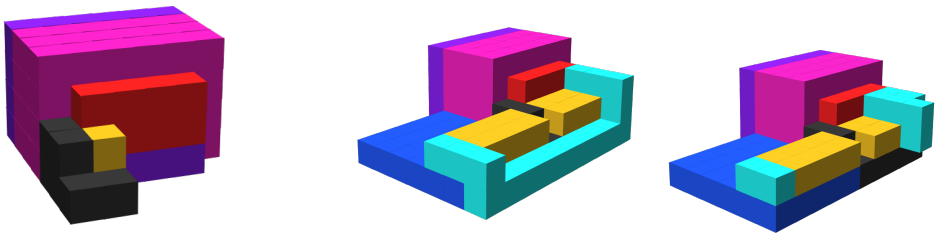
Whenever the filler verification process is completed on a surface of the bounding prism, signals activate glues on the corner gadgets of that surface which initiate the growth of an outer shell (see Figure 5). This outer shell consists of a rectangle of tiles which grows along the edge tiles of the inner shell underneath. Between these outer shell rectangles on each face of a bounding box, special corner gadgets called *outer corner gadgets* bind with 3 outer shell tiles on the corners of the assembly. Once an outer corner gadget attaches, signals are propagated along outer shell and outer edge tiles to adjacent outer corner gadgets. When enough of these signals are detected, the inner shell underneath the outer shell is dissolved and glues on the corners of the bounding box assembly called *candidate glues* are activated. These glues are complementary to glues presented on the inside corners of the outer shell. Because the inner shell dissolved, the bounding box assembly is free to move around inside of the outer shell, but since it is too big to fit through the gaps on the outer shell, it cannot escape. Additionally, since the outer shell is 1 tile bigger than the bounding box on each side, even though all corners of the two are complementary, only one of them will be able to bind. When one nondeterministically does, it is elected "leader" and the outer shell is dissolved.

3.2 Shape encoding

Following the process of leader election on a bounding box, we are presented with a single corner with unique glues exposed indicating a leader tile. Here we describe the tiles of E which allow for the universal shape encoding function f_e to be implemented on the shape contained in a bounding box. We use the term *voxels* to reference the locations of $\mathbb{Z}^3$ in the bounding box.

At a high level, the encoding of a shape $s_i \in S$ is generated by a process which visits each voxel inside of the bounding box sequentially, and transfers the information of whether the voxel is inhabited by a filler tile or a shape tile to an encoding assembly. Because leader election chooses a corner non-deterministically, there may be multiple encoding assemblies corresponding to each shape. We say that Φ_i is the finite set of encodings of shape s_i and define $\Phi = \Phi_1 \cup \cdots \cup \Phi_n$. For convenience we use ϕ_i to refer to any one encoding of s_i in Φ_i and note that the process is essentially identical for the others.

The first step in the process is for an *encoding corner gadget* to bind to the corner elected as leader, and then construct a set of helper tiles around the bounding box called the *shell*. Deconstruction is then carried out in *slices*, where each slice is the set of voxels contained in a 2D plane of the bounding box. The starting voxel contains the tile elected leader (see Figure 6a) and the orientation of the binding of the encoding corner gadget arbitrarily defines the orientation of the slices. For ease of explanation, once an orientation has been chosen by the attachment of the encoding corner gadget, we choose the x and y directions to correspond to the axes along a slice and the z direction to be the axis perpendicular to x and y into the bounding box from the leader. Specifically, each xy plane of the bounding box constitutes one slice. The end result of the encoding process is a rectangular prism assembly of height 1 where the each tile corresponds to a unique location of the bounding box in $\mathbb{Z}^3$, and whose glues represent whether or not each location contains a shape tile (represented by a 1), or empty space inhabited by a filler tile or otherwise (represented by 0). Additionally, information about the order in which tiles were deconstructed is included in ϕ_i for purposes of decoding and defining the width of a *row*. For a full description of the tiles and their functionality, see [4], and see Figure 6 for a schematic depiction of part of the process.



(a) The binding of the encoding gadget (black) allows for creation of the shell (fuchsia, purple) around the bounding box (red). Once the shell is complete, deconstruction can begin by the addition of recognizer tiles (yellow).

(b) (Left) Placement of recognizer tile allows for growth of path of tiles which extend the encoding structure and place both direction and encoding tile (Right) After tiles are placed on the encoding structure, tiles used to place the encoding tiles are dissolved into size 1 junk and the encoding process continues by placing a new recognizer in the order of encoding.

Figure 6 Tile type calculation and description of tile types used in shape decoding.

We provide a description of how the information provided by the location of tiles in a bounding box is encoded into binary values. Beginning with the origin point $(0, 0, 0)$, we read the tile type information for each tile in the first row sequentially by incrementing the x-coordinate; for example, the second tile read is in the voxel with coordinates $(1, 0, 0)$. Once all tiles in the current row have been read, we jump to the next row up. By this process of visiting every tile in a slice in a "zig-zag" pattern, we are able to encode the information regarding any slice of a bounding box sequentially. Figure 7 demonstrates the encoding of a $3 \times 4 \times 5$ $(x \times y \times z)$ bounding box.

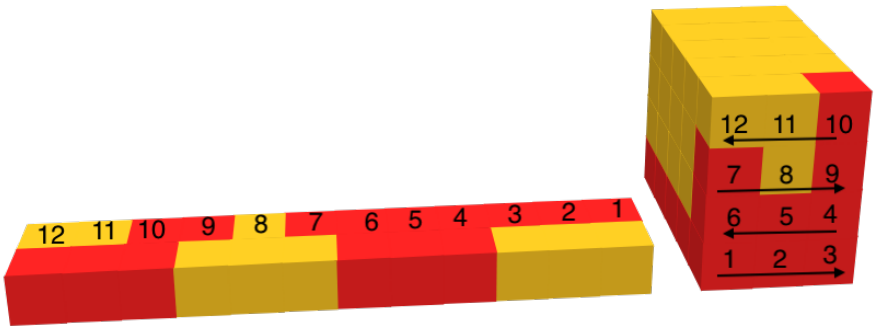


Figure 7 (Right) An example $3 \times 4 \times 5$ shape, (Left) The first two rows of its encoding assembly. The first (closest) row encodes the direction followed for each row of a slice, and the second row encodes the presence of a shape tile or filler tile in each location. Yellow tiles represent '0', red tiles represent "1". Shape tiles and '+' direction growth are encoded as 0, fill tiles and '-' are encoded as 1. The encodings of additional slices only need a single row each, since the growth direction is shared across rows of consecutive slices.

The very first row of the encoding subassembly contains the information regarding the direction of the growth in our zig-zag pattern, and as a byproduct we also are able to easily retrieve the width of the rows of tiles. We compare the x values in the coordinates (x, y, z) between the first tile of a row and the last tile of a row by subtracting the x value between the two such that $\Delta x = x_{\text{last}} - x_{\text{first}}$. If a tile is contained in a row where $\Delta x > 0$ we denote this growth in the positive ('+') direction. Alternatively, if $\Delta x < 0$ we denote this growth in the negative ('-') direction.

The STAM^R system $\mathcal{E}_S = (E, \Sigma_S, \tau = 2)$ generates the set of encoding assemblies Φ with a junk size of 4. $\mathcal{E}_S$ finitely completes, as each of the sub-constructions to carry out the encoding f_e require a finite number of steps (and thus, finite tile count) to complete. The final property which must hold is that regardless of the number of distinct shapes of input assemblies, the shapes of all will be correctly replicated. By our construction, there are never exposed glues on the surfaces of any pair of assemblies that each contain an input assembly that would allow them to bind to each other. Since junk assemblies produced by any assembly sequence are also unable to negatively interact with other assemblies, a system whose input assemblies have multiple shapes will behave simply as the union of individual systems which each have one input assembly shape, creating terminal assemblies of all of (and only) the correct shapes. Thus we prove Lemma 14 (with full details in [4]).

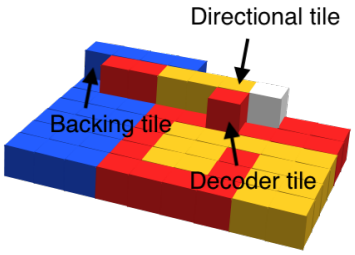
3.3 Shape Decoding

We now describe the tileset D which functions as a universal shape decoder. The STAM^R system for shape decoding is defined as $\mathcal{D}_\Phi = \{D, \Sigma_\Phi, \tau = 2\}$. Σ_Φ includes infinite copies of the tiles of D and the set Φ of encoding structures generated from $\mathcal{E}_S$. The shape decoding process and tile types required can be broken into 3 main sets of tiles. We describe the process for a single $\phi \in \Phi$ and note that the process proceeds identically for each encoding simultaneously. First, base tiles initiate the decoding process by binding to ϕ at a unique starting location. Base tiles also provide the remaining tile types to build an assembly which is guaranteed to be connected by at least strength 2 to the encoding structure. Second,

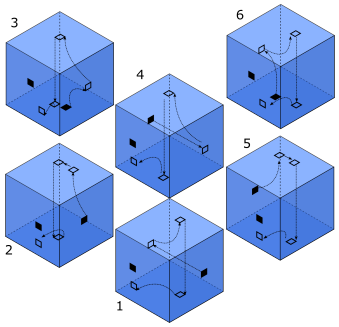
we construct the shape and filler tiles and describe how encoded information allows for an assembly sequence of shape tiles guaranteed to be connected to their neighbors in the encoded shape. Third, we have a set of tiles which read the encoding of filler and shape tiles (decoder tiles), and allow for the sequential placement of shape and filler tiles based on their location in the encoding of a shape. In the tileset D, we use a decoding process which places tiles in the exact same order as the encoding process built the encoding assembly ϕ as described in Figure 7. For a full description of the tiles and their functionality, see [4].

3.3.1 Retrieving Encoded Information

Two pieces of information are explicitly encoded in ϕ. The bulk of the tiles in the encoding correspond to identifying if a location in a shape corresponds to empty space, or a tile of the shape. The second piece of information provided in the first row of the encoding is the the direction of growth; this can be utilized in two manners. First, the direction of growth provides to the system the types of tiles to be utilized to reach the point encoded, as growth processes vary significantly between "+" direction growth (encoded as a 0) and "-" direction growth (encoded as a "1"). Secondly, when the direction of growth encoded changes from 0 to 1 or 1 to 0, this indicates to the system when a tile is to be placed into a new row. This information is required to ensure that we can grow a slice such that each tile is guaranteed to be connected to its neighbor, and so tile faces are assigned the appropriate glues. Figure 8a demonstrates the tile bindings allowing information to be retrieved from an encoding.



(a) An example of the information which is gathered from the encoding structure. The directional tile gathers information regarding the growth type of tile location encoded. The direction change detector gadget (white) which detects that growth type '0' shifts to growth type '1', indicating a change of row and necessitating a direction change tile. The decoder tile, once glues are available to cooperatively bind to the encoding structure and the directional tile, determines that the tile in the current location is a shape tile. Backing tiles allow for the sensing of the first row.



(b) An example of normal row growth and direction change tiles used by the decoding process to build a slice. These tile types map to both shape and fill tiles. (1) and (4) are standard row growth tiles for '+' and '-' direction growth, respectively. (2) and (5) are row end tiles for '+' and '-' direction growth; they open cooperative binding sites which allow for tiles (3) and (6) to bind and change the direction of growth. Signal activation arrows demonstrate the order in which faces of shape tiles are determined to be either bound to a neighboring shape tile or have a fill tile adjacent to the face.

Figure 8 Tile type calculation and description of tile types used in shape decoding.

3.3.2 Proof of Universal Shape Decoding Correctness

We summarize the decoding process and show that during this process, the shapes which were encoded in the set of input encoding assemblies Φ are correctly assembled. We first consider the decoding process of a single encoding assembly $\phi \in \Phi$ and note that a similar process happens for all encoding assemblies simultaneously without interfering one another.

Our decoding process begins by building a base of tiles connected to ϕ. This base holds the shape as it is being constructed and is used to help ensure the connectivity of the shape as it is being constructed. The decoding process is performed in iterations, where during each iteration a row of ϕ is scanned tile-by-tile and a corresponding 2D slice of the shape is constructed. Each slice is constructed starting from the bottom (smallest y coordinate) to the top (largest y coordinate), with tiles attaching in a zig-zag manner, as illustrated in Figure 7. Each slice of the assembled shape corresponds to a unique z coordinate so for convenience we call the slice whose z coordinate is i, σ_i. As each slice is assembled, tiles are placed in each location of the slice, even those locations that will not be part of the final shape, though these will be removed during the assembly of the next slice.

The first slice σ_1 can be assembled naively, but during the assembly of each following slice, tiles which will not be part of the final shape on the previous slice must be removed. This is done as follows. Suppose that slice σ_i $(i > 1)$ is currently being assembled. Before a tile t_i is placed in a location (x, y, i), a gadget is used to determine the type of the tile (see Figure 8b for examples of the 6 main tile types) t_{i-1} at location $(x, y, i-1)$ (i.e. the tile with the same x and y coordinates on the previous slice). If this t_{i-1} is part of the final shape, then t_i is placed and signals are used to activate strength 2 glues between t_i and t_{i-1}; otherwise, if t_{i-1} is not part of the final shape, it is removed before t_i is placed. Regardless of the type of tile t_{i-1}, when t_i is placed, glues are activated which connect t_i to all adjacent tiles on the same slice. Once the final slice is assembled, a final zig-zag pass is made in the next z coordinate to remove all tiles from the last slice which are not part of the final shape.

It is also important to note that the base, on which the shape is being assembled, also forms a ceiling above the slices being assembled. This ceiling helps ensure that tiles on the top row of each slice are able to remain attached to the assembly during construction. It should be clear that during this decoding process (1) each tile that belongs to the final shape is placed in its correct location, and (2) that those tiles of a slice which are not part of the final shape will be removed from the assembly during the assembly of the next slice; However, because tiles are removed during the process, we must show that none of these removals can cause parts of the assembly to unintentionally detach. We state this as Lemma 16. The proof of this can be found in [4]

▶ **Lemma 16.** *Let ϕ be an encoding assembly which encodes the shape s. During the decoding process above, as slice σ_i $(i > 1)$ is being assembled, no tile in slices $\sigma_1, \ldots, \sigma_{i-1}$ which are part of the final shape assembly can detach.*

Given the set of input encoding structures Φ, the STAM^R system $\mathcal{D}_\Phi = \{D, \Sigma_\Phi, \tau = 2\}$ produces a set of terminal supertiles $S = \{s_1, \ldots, s_n\}$ in parallel with a maximum junk size of 3. $\mathcal{D}_\Phi$ finitely completes, as for the production of the set of shapes $s \in S$ from input encoding structures Φ a finite number of tiles are required for each encoding structure to produce a terminal assembly. We can guarantee this as each encoding produces a single terminal shape, as the encoding of the shape dissolves into size 1 junk after the terminal shape has decoded. By our construction, there are never exposed glues on the surfaces of any pair of assemblies that each contain an input encoding that would allow them to bind to each other. Since junk assemblies produced by any assembly sequence are also unable to interact with other assemblies, a system whose input assemblies have multiple shapes will behave simply as the union of individual systems which each have one input assembly shape, creating terminal assemblies of all of (and only) the correct shapes. Thus we prove Lemma 15.

Since we have shown the existence of universal encoding and universal decoding tilesets, we have the basis to demonstrate a universal shape replicator and prove Theorem 13. Due to the modularity of the constructions of Lemmas 14 and 15, at a high level we simply generate a new tileset $R = E \cup D$. A few minor changes must occur to satisfy Definition 12; the full proof can be found in [4].

4 Beyond Shape Replication

The constructions used to prove Theorem 13 were intentionally broken into separate, modular constructions proving Lemmas 14 and 15 and thus providing a universal shape encoder and a universal shape decoder. This is not only useful for proving their correctness, but also for allowing for computational transformations to be performed on the encodings of input shapes in order to instead produce output shapes based on those transformations. Like even the much simpler aTAM, the STAM (and STAM^R) are Turing universal, meaning any arbitrary computer program can be executed by systems in these models. Thus, given any program that can perform a computational transformation of the points of a shape and output points of another shape, tiles that execute that program (for instance, by simulating an arbitrary Turing machine in standard ways, e.g. [30, 24]) can receive as input the binary encodings of arbitrary shapes (after their creation by the universal shape encoder), transform them in any algorithmic manner, and then assemblies of the shapes output by those transformations can be produced (using the universal shape decoder).

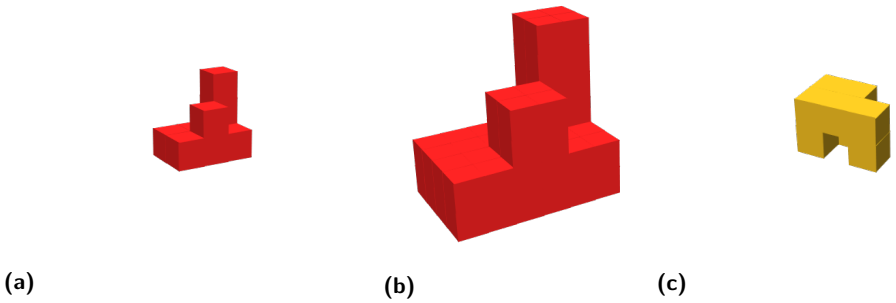


Figure 9 (a) An example shape, (b) The same shape at scale factor 2, (c) A shape which is complementary to the top surface of the shape in (a).

Due to space constraints, we do not go into great detail about the opportunities that such constructions provide. Instead, we mention just a few of the possibilities (and depict some in Figure 9) while noting that the possibilities are technically infinite:

1. Scaled shapes: a system could be designed to produce assemblies that have the shapes of input assemblies scaled by either a built-in constant factor (including negative, to shrink the shapes), or instead with another type of input assembly that specifies the scaling factor, allowing for a "universal shape scaler".
2. Inverse shapes: a system could be designed to produce assemblies that have the inverse, i.e. complementary, shapes of input assemblies (assuming the complements are connected, and restricting to a bounding box size since the complement of any finite shape is infinite).
3. Pattern matching: a system could be designed to inspect input assembly shapes for specific patterns and to either produce assemblies that signal the presence of a target pattern, or instead assemblies that are complementary to, and can bind to, the surfaces of assemblies containing those patterns.

Although such constructions are highly theoretical and quite complex, and thus unlikely in their current forms to be practically implementable, they provide a mathematical foundation for the construction of complex, dynamic systems that mimic biological systems. One possible example is an "artificial immune system" capable of inspecting surfaces, detecting those which match (or fail to match) specific patterns, and creating assemblies capable of binding to those deemed to be foreign, harmful, or otherwise targeted. As mentioned, there are infinite possibilities.

5 Impossibility of Shape Replication Without Deconstruction

In this section, we prove that in order for a system in the STAM^R to encode and/or replicate shapes which have enclosed or bent cavities (see Definitions 4 and 5), the input assemblies must have the potential for tiles to be removed. To do so, we first utilize a theorem from [2].

▶ **Theorem 4** (from [2]). *Let U be an STAM* tileset such that for an arbitrary 3D shape S, the STAM* system $\mathcal{T} = (U, \sigma_S, \tau)$ with* dom $\sigma_S = S$*, $\mathcal{T}$ is a shape self-replicator for S and σ_S is non-porous. Then, for any $r \in \mathbb{N}$, there exists a shape S such that $\mathcal{T}$ must remove at least r tiles from the seed assembly σ_S.*

Theorem 4 from [2] applies to the STAM* (see Section A.3). However, the STAM^R is simply a restricted version of the STAM* which only allows tiles to be a single shape, that of a unit cube, and does not allow flexible glues. Since all assemblies in the STAM^R are non-porous (i.e. free tiles cannot pass through the tiles of an assembly or the gaps between bound tiles) and the STAM^R has more restrictive dynamics than the STAM*, the proof of this result, which shows the impossibility of self-replicating assemblies with enclosed cavities without removing tiles, suffices to prove the following corollary (stated using the terminology of this paper) as well.[2] Note that this proof holds even if an input assembly is not uniformly covered, but its glues are instead allowed to encode information about the shape.

▶ **Corollary 17.** *There exist neither a universal shape encoder nor a universal shape replicator in the STAM^R for the class of shapes with enclosed cavities whose assemblies are not deconstructable.*

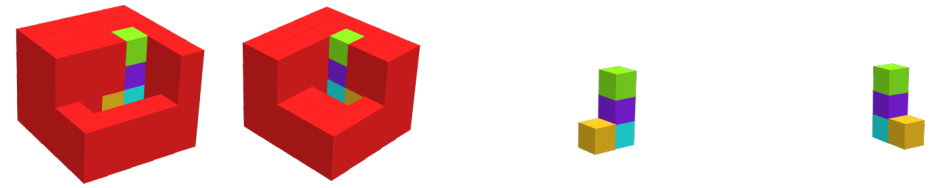


Figure 10 (a) and (b) Partial depictions of a pair of shapes which cannot be correctly encoded/replicated without a deconstructable input assembly. Each consists of a $5 \times 5 \times 4$ cube with a 4-cube-long bent cavity. For each, the green, purple, blue, and yellow locations indicate the empty locations that make the bent cavity. The rest of the $5 \times 5 \times 4$ cube locations would be filled in with red cubes (some have been omitted to make the cavity locations visible). (c) and (d) The shapes of assemblies that could grow into the bent cavities.

Our next theorem deals with shapes having bent cavities.

[2] The proof can be found in [2], and we omit duplicating it here due to space constraints.

▶ **Theorem 18.** *There exist neither a universal shape encoder nor a universal shape replicator in the STAM^R for the class of shapes with bent cavities whose input assemblies are uniformly covered but are not deconstructable.*

We prove Theorem 18 by contradiction. Therefore, let f_e be a shape encoding function and assume E is a universal shape encoder with respect to f_e, and let c be the constant value which bounds the size of the junk assemblies. (Nearly identical arguments will hold for a universal shape replicator.) Define the shapes s_1 and s_2 as shown in Figures 10a and 10b, i.e. each is a $5 \times 5 \times 4$ cube with a bent cavity that goes into the cube to a depth of 3, then turns one of two directions for each. Note importantly that the well is offset from the center of the cube such that s_1 and s_2 are not rotationally equivalent. Since E is assumed to be a universal shape encoder, there must exist two STAM^R systems $\mathcal{E}_1 = (E, \sigma_1, \tau)$ and $\mathcal{E}_2 = (E, \sigma_2, \tau)$, where σ_1 consists of infinite copies of tiles from E and infinite copies of uniformly covered assemblies in the shape of s_1, and σ_2 consists of infinite copies of tiles from E and infinite copies of uniformly covered assemblies in the shape of s_2 as follows.

$\mathcal{E}_1$ must produce terminal assemblies which encode shape s_1 but must not produce terminal assemblies which encode shape s_2, since no assembly of shape s_2 is included in its input assemblies. Similarly, $\mathcal{E}_2$ must produce terminal assemblies which encode shape s_2 but not s_1. Let $\vec{\alpha}$ be an assembly sequence in $\mathcal{E}_1$ which results in a terminal assembly encoding shape s_1. We now show that every action of $\vec{\alpha}$ must be valid, in the same ordering, in $\mathcal{E}_2$ but using an input assembly of shape s_2. This is because the exact same glues will be exposed by the input assemblies of shapes s_1 and s_2 in the same relative locations with the slight difference of relative rotations of the innermost locations of the bent cavities of each from the adjacent cavity locations. Assuming that, in $\vec{\alpha}$, tiles attach into all locations of the bent cavity (if only the location shown in yellow remains empty the same argument will hold, and if both the locations shown in yellow and blue remain empty then there is absolutely no difference in any aspect of the assembly sequence in $\mathcal{E}_2$ and the argument immediately holds), this results only in the relative orientations of at most the bottom two tiles being turned 90 degrees relative to the tile immediately above them (i.e. the tile in the purple location in Figure 10). Since tiles in the STAM^R are rotatable, with no distinction for directions, there is no mechanism for tiles in the purple locations of assemblies shown in Figures 10c and 10d from distinguishing from each other (via tile types, glues, or signals). Tiles of the same types which bind into those locations in $\vec{\alpha}$ must also be able to do so in the assembly sequence of $\mathcal{E}_2$ using the exact same glues and firing the exact same signals (if any). Thus $\vec{\alpha}$ must be a valid assembly sequence in $\mathcal{E}_2$ as well. This means that an assembly encoding the shape of s_1 is also created as a terminal assembly in $\mathcal{E}_2$. Note that if the constant c is greater than the size of the shapes s_1 and s_2 (i.e. $5 * 5 * 4 - 4 = 96$), then we can simply increase their dimensions until they are larger than c (but still contain the same bent cavities) and the argument still holds and the incorrectly produced assemblies cannot be considered "junk" assemblies. This is a contradiction that E is a universal shape encoder with respect to f_e and constant c. Since no assumptions were made about E other than it being a universal shape encoder, no such E can exist. By slightly altering the argument for a universal shape replicator R, and generating terminal assemblies of shapes s_1 and s_2 (rather than their encodings), the same argument holds to show no universal shape replicator exists. Thus Theorem 18 is proven.

References

1 Zachary Abel, Nadia Benbernou, Mirela Damian, Erik D. Demaine, Martin L. Demaine, Robin Flatland, Scott D. Kominers, and Robert T. Schweller. Shape replication through self-assembly and RNAse enzymes. In *SODA 2010: Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1045–1064, Austin, Texas, 2010. Society for Industrial and Applied Mathematics.

2 Andrew Alseth, Daniel Hader, and Matthew J. Patitz. Self-replication via tile self-assembly. Technical Report 2105.02914, Computing Research Repository, 2021. `arXiv:2105.02914`.

3 Andrew Alseth, Daniel Hader, and Matthew J. Patitz. Self-Replication via Tile Self-Assembly (Extended Abstract). In Matthew R. Lakin and Petr Šulc, editors, *27th International Conference on DNA Computing and Molecular Programming (DNA 27)*, volume 205 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:22, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.DNA.27.3`.

4 Andrew Alseth, Daniel Hader, and Matthew J. Patitz. Universal shape replication via self-assembly with signal-passing tiles, 2022. `doi:10.48550/ARXIV.2206.03908`.

5 Sarah Cannon, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Matthew J. Patitz, Robert T. Schweller, Scott M. Summers, and Andrew Winslow. Two hands are better than one (up to constant factors): Self-assembly in the 2HAM vs. aTAM. In Natacha Portier and Thomas Wilke, editors, *STACS*, volume 20 of *LIPIcs*, pages 172–184. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

6 Cameron Chalk, Erik D. Demaine, Martin L. Demaine, Eric Martinez, Robert Schweller, Luis Vega, and Tim Wylie. Universal shape replicators via Self-Assembly with Attractive and Repulsive Forces. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 225–238. Society for Industrial and Applied Mathematics, January 2017. `doi:10.1137/1.9781611974782.15`.

7 Qi Cheng, Gagan Aggarwal, Michael H. Goldwasser, Ming-Yang Kao, Robert T. Schweller, and Pablo Moisset de Espanés. Complexities for generalized models of self-assembly. *SIAM Journal on Computing*, 34:1493–1515, 2005.

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

9 Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T. Schweller, and Diane L. Souvaine. Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues. *Natural Computing*, 7(3):347–370, 2008. `doi:10.1007/s11047-008-9073-0`.

10 Erik D. Demaine, Matthew J. Patitz, Trent A. Rogers, Robert T. Schweller, Scott M. Summers, and Damien Woods. The two-handed assembly model is not intrinsically universal. In *40th International Colloquium on Automata, Languages and Programming, ICALP 2013, Riga, Latvia, July 8-12, 2013*, Lecture Notes in Computer Science. Springer, 2013.

11 Erik D. Demaine, Matthew J. Patitz, Robert T. Schweller, and Scott M. Summers. Self-Assembly of Arbitrary Shapes Using RNAse Enzymes: Meeting the Kolmogorov Bound with Small Scale Factor (extended abstract). In Thomas Schwentick and Christoph Dürr, editors, *28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, volume 9 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 201–212, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.STACS.2011.201`.

12 David Doty, Jack H. Lutz, Matthew J. Patitz, Robert T. Schweller, Scott M. Summers, and Damien Woods. The tile assembly model is intrinsically universal. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science*, FOCS 2012, pages 302–310, 2012.

13 Jérôme Durand-Lose, Jacob Hendricks, Matthew J. Patitz, Ian Perkins, and Michael Sharp. Self-assembly of 3-D structures using 2-D folding tiles. In David Doty and Hendrik Dietz, editors, *DNA Computing and Molecular Programming - 24th International Conference, DNA 24, Jinan, China, October 8-12, 2018, Proceedings*, volume 11145 of *Lecture Notes in Computer Science*, pages 105–121. Springer, 2018.

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

15 Tyler Fochtman, Jacob Hendricks, Jennifer E. Padilla, Matthew J. Patitz, and Trent A. Rogers. Signal transmission across tile assemblies: 3D static tiles simulate active self-assembly by 2D signal-passing tiles. *Natural Computing*, 14(2):251–264, 2015.

16 Jacob Hendricks, Meagan Olsen, Matthew J. Patitz, Trent A. Rogers, and Hadley Thomas. Hierarchical self-assembly of fractals with signal-passing tiles. Submit to Natrual Computing.

17 Jacob Hendricks, Matthew J. Patitz, and Trent A. Rogers. Replication of arbitrary hole-free shapes via self-assembly with signal-passing tiles. In Cristian S. Calude and Michael J. Dinneen, editors, *Unconventional Computation and Natural Computation - 14th International Conference, UCNC 2015, Auckland, New Zealand, August 30 - September 3, 2015, Proceedings*, volume 9252 of *Lecture Notes in Computer Science*, pages 202–214. Springer, 2015. doi:10.1007/978-3-319-21819-9_15.

18 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, pages 800–809, 2016.

19 Jacob Hendricks, Matthew J. Patitz, and Trent A. Rogers. Reflections on tiles (in self-assembly). *Natural Computing*, 16(2):295–316, 2017. doi:10.1007/s11047-017-9617-2.

20 Nataša Jonoska and Daria Karpenko. Active tile self-assembly, Part 1: Universality at temperature 1. *International Journal of Foundations of Computer Science*, 25(02):141–163, 2014. doi:10.1142/S0129054114500087.

21 Nataša Jonoska and Daria Karpenko. Active tile self-assembly, Part 2: Self-similar structures and structural recursion. *International Journal of Foundations of Computer Science*, 25(02):165–194, 2014. doi:10.1142/S0129054114500099.

22 Yonggang Ke, Luvena L Ong, William M Shih, and Peng Yin. Three-dimensional structures self-assembled from DNA bricks. *Science*, 338(6111):1177–1183, 2012.

23 Alexandra Keenan, Robert T. Schweller, and Xingsi Zhong. Exponential replication of patterns in the signal tile assembly model. In David Soloveichik and Bernard Yurke, editors, *DNA*, volume 8141 of *Lecture Notes in Computer Science*, pages 118–132. Springer, 2013. doi:10.1007/978-3-319-01928-4_9.

24 James I. Lathrop, Jack H. Lutz, Matthew J. Patitz, and Scott M. Summers. Computability and complexity in self-assembly. *Theory Comput. Syst.*, 48(3):617–647, 2011. doi:10.1007/s00224-010-9252-0.

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

26 Austin Luchsinger, Robert Schweller, and Tim Wylie. Self-assembly of shapes at constant scale using repulsive forces. *Natural Computing*, August 2018. doi:10.1007/s11047-018-9707-9.

27 Pierre-Étienne Meunier, Damien Regnault, and Damien Woods. The program-size complexity of self-assembled paths. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 727–737. ACM, 2020. doi:10.1145/3357713.3384263.

28 Jennifer E. Padilla, Matthew J. Patitz, Robert T. Schweller, Nadrian C. Seeman, Scott M. Summers, and Xingsi Zhong. Asynchronous signal passing for tile self-assembly: Fuel efficient computation and efficient assembly of shapes. *International Journal of Foundations of Computer Science*, 25(4):459–488, 2014.

29 Matthew J. Patitz, Robert T. Schweller, and Scott M. Summers. Exact shapes and Turing universality at temperature 1 with a single negative glue. In Luca Cardelli and William M. Shih, editors, *DNA Computing and Molecular Programming - 17th International Conference, DNA 17, Pasadena, CA, USA, September 19-23, 2011. Proceedings*, volume 6937 of *Lecture Notes in Computer Science*, pages 175–189. Springer, 2011.

30 Matthew J. Patitz and Scott M. Summers. Self-assembly of decidable sets. *Natural Computing*, 10(2):853–877, 2011. doi:10.1007/s11047-010-9218-9.

31 Matthew J. Patitz and Scott M. Summers. Identifying shapes using self-assembly. *Algorithmica*, 64(3):481–510, 2012.

32 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*, pages 459–468, Portland, Oregon, United States, 2000. ACM.

33 Rebecca Schulman, Bernard Yurke, and Erik Winfree. Robust self-replication of combinatorial information via crystal growth and scission. *Proceedings of the National Academy of Sciences*, 109(17):6405–10, 2012. URL: `http://www.biomedsearch.com/nih/Robust-self-replication-combinatorial-information/22493232.html`.

34 David Soloveichik and Erik Winfree. Complexity of self-assembled shapes. *SIAM Journal on Computing*, 36(6):1544–1569, 2007. `doi:10.1137/S0097539704446712`.

35 Scott M. Summers. Reducing tile complexity for the self-assembly of scaled shapes through temperature programming. *Algorithmica*, 63(1-2):117–136, June 2012. `doi:10.1007/s00453-011-9522-5`.

36 Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.

37 Damien Woods, David Doty, Cameron Myhrvold, Joy Hui, Felix Zhou, Peng Yin, and Erik Winfree. Diverse and robust molecular algorithms using reprogrammable DNA self-assembly. *Nature*, 567:366–372, 2019.

A Technical Appendix

A.1 Definition of the STAM^R model

Here we provide a definition of the model used in this paper, called the STAM^R (i.e. the "STAM with rotation"), which is based upon the 3D Signal-passing Tile Assembly Model (STAM) [15] (and similar to the model in [20, 21]). The STAM is itself based upon the 2-Handed Assembly Model (2HAM) [7, 9], also referred to as the "Hierarchical Assembly Model", which is a mathematical model of tile-based self-assembling systems in which arbitrarily large pairs of assemblies can combine to form new assemblies.

A *glue* is an ordered pair (l, s), where $l \in \Sigma^+ \cup \{s^* : s \in \Sigma^+\}$ is a non-empty string, called the *label*, over some alphabet Σ, possibly concatenated with the symbol "$*$", and $s \in \mathbb{Z}^+$ is a positive integer, called the *strength*. A glue label l is said to be *complementary* to the glue label l^*.

A *tile type* is a mapping of zero or more glues, along with *glue states* and possibly *signals*, which will be defined shortly, to the 6 faces of a unit cube. A *tile* is an instance of a tile type, and is the base component of the STAM^R. Each tile type is defined in a canonical orientation, but tiles can be in that orientation or any rotation which is orthogonal to it (i.e. they are embedded in $\mathbb{Z}^3$).

Every glue can be in one of three *glue states*: $\{\texttt{on}, \texttt{latent}, \texttt{off}\}$. If two tiles are placed next to each other, and their adjacent faces have glues $g_1 = (l, s)$ and $g_2 = (l^*, s)$, then those glues can form a *bond* whose *strength* is s. We require any copies of glues with the label l, or its complement l^*, in any given system have the same strength (e.g. it is not allowed to have one glue labeled l with strength 1 and another labeled l or l^* with strength 2).

A *signal* is a mapping from a glue g_s (the *source glue*) to an ordered pair, (g_t, s), where g_t (the *target glue*) is a glue on the same tile as g_s (possibly g_s itself) and $s \in \{\texttt{on}, \texttt{off}\}$. If and when g_s forms a bond with its complementary glue on an adjacent tile, the signal is *fired* to change the state of g_t to state s. Each glue of a tile type can be defined to have zero or more signals assigned to it. Each signal on a tile can fire at most a single time. When a glue is fired, the state of the target glue is not immediately changed, but the pair (g_t, s) is

added to a queue of *pending signals* for the tile containing its glues. When a pending glue is selected for completion (in a process described below), then the state of g_t is changed to s if and only if its current state is s_0 and $(s_0, s) \in \{(\texttt{on}, \texttt{off}), (\texttt{latent}, \texttt{on}), (\texttt{latent}, \texttt{off})\}$. That is, the only valid glue state transitions are `on` to `off`, or `latent` to `on` or `off`.

A *supertile* is (the set of all translations and rotations of) a positioning of one or more connected tiles on the integer lattice $\mathbb{Z}^3$. Two adjacent tiles in a supertile can form a bond if the glues on their abutting sides are complementary and both are in the `on` state. Each supertile induces a *binding graph*, a grid graph whose vertices are tiles, with an edge between every pair of bound tiles whose weight is the strength of the bound glues. A supertile is *τ-stable* if every cut of its binding graph cuts edges whose weights sum to at least τ. That is, the supertile is τ-stable if at least energy τ is required to separate the supertile into two parts. *Assembly* is another term for a supertile, and we use the terms interchangeably, to mean the same thing.

Each tile has a *tile state* that contains the current state of every glue as well as a (possibly empty) set of pending signals and a (possibly empty) set of completed signals. Every supertile consists of not only its set of constituent tiles, but also their tile states, and a set bonds that have formed between pairs of glues on adjacent tiles.

A system in the STAM^R is an ordered triple (T, S, τ) where T is a finite set of tiles called the *tileset*, S is a *system state* which consists of a multiset of supertiles that each have a count (possibly infinite), and $\tau \in \mathbb{Z}^+$ is the *binding threshold* (a.k.a. *temperature*) parameter of the system which specifies the minimum strength of bonds needs to hold a supertile together. In the initial state of a system, no tiles have pending signals, all pairs of adjacent glues which are both complementary and in the `on` state in all supertiles have formed bonds and any signals which would have been fired by those bonds are completed, and all distinct supertiles are assumed to start arbitrarily far from each other (i.e. none is enclosed within another). By default (and unless otherwise specified), the initial state contains an infinite count of all singleton tiles in T.

A system evolves as a (possibly infinite) series of discrete steps, called an *assembly sequence*, beginning from its initial state. Each step occurs by the random selection and execution of one of the following actions:

1. Two supertiles currently in the system, α and β, are translated and/or rotated without ever overlapping so that they can form bonds whose strengths sum to at least τ. The count of the newly formed supertile is increased by 1 in the system state and the counts of each of α and β are decreased by 1 (if they aren't ∞). In the newly created supertile, from the entire set of pairs of glues which can form bonds, a random subset whose strengths sum to $\geq \tau$ is selected and bonds formed by those glues are added to the set of bonds that have formed for that supertile. Additionally, for each glue which forms a bond, all signals for which it is a source glue, but which aren't already pending or completed, are added to the set of pending signals for its tile.
2. For any supertile currently in the system, from the set of pairs of glues which can form bonds but haven't, a glue pair is selected and a bond formed by those glues is added to the set of bonds that have formed for that supertile. Additionally, for each glue which forms that bond, all signals for which it is a source glue, but which aren't already pending or completed, are added to the set of pending signals for its tile.
3. For any supertile currently in the system, a pending signal is selected from the set of pending signals of one of its tiles. If the action specified by that signal is valid, the state of the target glue is changed to the state specified by the signal. The signal is removed

from the set of pending signals and added to the set of completed signals. If the action is not valid (i.e. the pair specifying the current state of the target glue and the desired end state is not in $\{(\texttt{on}, \texttt{off}), (\texttt{latent}, \texttt{on}), (\texttt{latent}, \texttt{off})\}$), then the signal is just removed from the pending set and added to the completed set, and there is no change to the target glue.

4. For a supertile γ currently in the system for which there exists one or more cuts of $< \tau$ (which could be the case due to one or more glues changing to the `off` state), one of those cuts is randomly selected and γ is split into two supertiles, α and β, along that cut. The count of γ in the system state is decreased by one (if it isn't ∞) and the counts of α and β are increased by one (if they aren't ∞).

Given a system $\mathcal{T} = (T, S, \tau)$, a supertile is *producible*, written as $\alpha \in \mathcal{A}[\mathcal{T}]$, if it either is contained in the initial state S or it can be formed, starting from S, by any series of the above steps. A supertile is *terminal*, written as $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$, if it is producible and none of the above actions are possible to perform with it (and any other producible assembly, for list item 1).

Note that tiles are not allowed to diffuse through each other, and therefore a pair of combining supertiles must be able to translate and/or rotate without ever overlapping into positions for binding. It is allowed, though, for two supertiles, α and β, to translate and/or rotate into locations which are partially enclosed by another supertile γ before binding, potentially creating a new supertile, δ, which would not have been able to translate and/or rotate into that location inside γ, without overlapping γ, after forming. However, although the model allows for supertiles to assemble "inside" of others, in order to strengthen our results we do not utilize it for the constructions of our positive results, but its possibility does not impact our negative result.

A.2 STAM^R Gadgets and Tools

Throughout our results we repeatedly make use of several small assemblies of tiles, referred to as *gadgets*, and patterns of signal activations to accomplish tasks such as keeping track of state, removing specific tiles, and passing information across an assembly. In this section we describe several of these gadgets and signal patterns so that they can later be referenced during our construction. We intend that this section also serve as a basic introduction by example to the dynamics of signal tile assembly.

Detector Gadgets

Detector gadgets are used to detect when a specific set of tiles exist in a particular configuration relative to one another in an assembly. For a detector gadget to work, the tiles to be detected need to each be presenting a glue unique to the configuration to be detected. The strength of these glues should add to at least the binding threshold τ, but the total strength of any proper subset of the glues should not. If two or more tiles then exist in the configuration expected by the detector gadget, the gadget can cooperatively bind with the relevant glues. Upon binding, any signals with the newly bonded glues as a source will fire. These signals can be in the "detected tiles" or in the detector itself and can be used to initiate some other process based on the condition that the tiles exist in the specified configuration. More often than not, it's also desirable for signals within the detector gadget to deactivate its own glues so that it does not remain attached to the assembly after the detection has occurred.

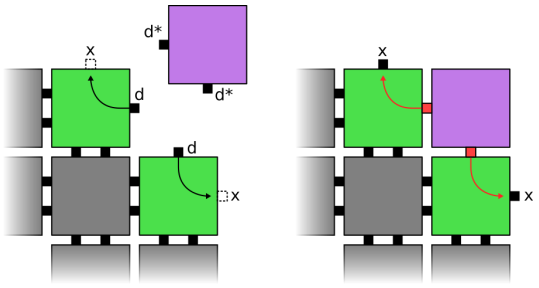


Figure 11 A simple detector gadget example.

Detector gadgets can exist in many forms depending on the configuration to detect, but the most simple is a single tile. Illustrated in Figure 11 is a simple detector gadget designed to detect 2 diagonally adjacent tiles, each presenting a strength-1 glue of type d towards a shared adjacent empty tile location. In this case, $\tau = 2$ and the detected tiles are designed to activate their x glues upon a successful detection. In general, detector gadgets can be made up of more than 1 tile. Duples of tiles can be used for instance to detect immediately adjacent tiles each presenting some specific glue on the same side. For detector gadgets consisting of more than 1 tile, the component tiles must be designed to have unique τ-strength glues between them so that the components can bind together piece-wise to form the whole gadget. Because all of the glues presented for the detection are needed to reach a cumulative strength of τ, only after it is fully formed will it be able to detect tiles and thus partially assembled detector gadgets will not erroneously perform partial detections. It is assumed in our results that signals within a detector gadget itself will cause the gadget to dissolve after a detection.

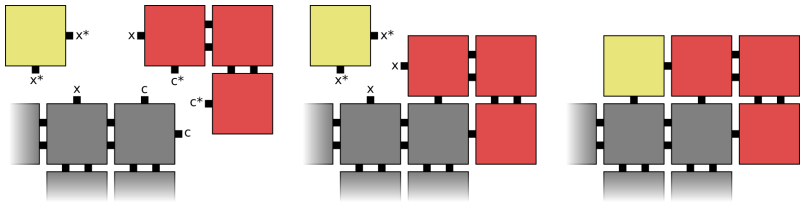


Figure 12 A corner gadget example.

Corner Gadgets

Corner gadgets are a specific type of detector gadget which are used primarily for facilitating the attachment of other tiles on the surface of some assembly. Corner gadgets can either be 2D, consisting of 3 tiles arranged in a 2×2 square with one corner missing, or 3D, consisting of 7 tiles arranged in a $2 \times 2 \times 2$ cube with one of the corners missing. Because of this shape, a corner gadget is able to cooperatively bind to any single tile of an assembly with 2 accessible, adjacent faces. These faces must be presenting specified glues whose cumulative strength is at least τ, but neither individually is. Illustrated in Figure 12 is the side view of a 2D corner gadget attaching to an assembly. After the attachment, it is then possible for additional tiles to cooperatively bind along the surface of the assembly. This behavior is useful for initiating the growth of shells of tiles around an assembly as will be seen in our later construction.

Like with detector gadgets, signals fired from the binding of a corner gadget can also be used to initiate other tasks, though special care needs to be taken for 3D corner gadgets when $\tau = 2$. Because a 3D corner gadget has 3 interior faces which can have glues to bind

with a tile on the corner of an assembly, it is often desirable to fire signals from all 3 of these glues; however, because only 2 glues are necessary to meet the binding threshold when $\tau = 2$, the third may not form a bond immediately. If it is planned for the corner gadget to eventually detach, then it is crucial that any signals causing the corner gadget to detach cannot fire until all 3 of the interior glues have first bound. This can often be accomplished using *sequential signaling* as described below.

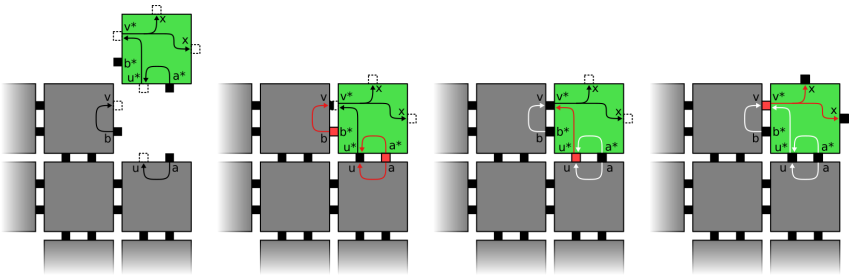


Figure 13 Sequential signaling example.

Sequential Signaling

By carefully adding additional helper glues and signals to a tile or tiles, we can ensure that signals in our tiles are fired in a specific order or ensure that a certain set of glues has successfully bound before certain signals are fired. The way in which this is done depends on the exact situation, but as an example consider the situation illustrated in Figure 13. In this situation we want the green tile to cooperatively bind to the assembly via glues of type a and b. Once this happens, we want to first activate additional glues of type u and v between the green tile and assembly so that each side of the green tile is attached to the assembly with strength 2, then we want glues of type x on the other sides of the green tile to activate. The arrangement of signals illustrated in Figure 13 guarantees that the x glues cannot activate before both the u and v glues do, since the signals which activate the x glues are dependent on the glues u and v. A similar arrangement of signals and glues is used to implement gadgets called *filler tiles* in our construction.

Tile Conversion

It is often useful for tiles to change behavior after receiving a specific signal. This can be done by having signals activate a new set of glues on the tile and deactivate old ones. This can be thought of as converting the tile into a different type of tile, but it's important to note that this process cannot happen indefinitely nor arbitrarily. Every tile conversion has to be prepared in the signals and latent glues of the tile and once those signals fire, they cannot fire again. It is possible for a tile to convert to another several times, but such a tile must have the necessary glues and signals for each conversion separately. It is also often possible achieve this behavior by detachment of one tile and attachment of another in the same location, though special care needs to be taken so that no other tiles can attach in the location during the conversion.

Tile Dissolving

For any arbitrary set of glues on a tile, we use the term *dissolving* to refer to the process of initiating signals which turn all possible glues to the `off` state We note that due to the asynchronous nature of the model that no guarantee can be made with regards to the order

of the processing of the signals. A tile breaks apart from its supertile once a strength τ bond no longer exists between itself and its neighbors. However, other glues may be active when the tile does so, leading to the possibility of undesired binding due to exposed glues in the `on` state with pending `off` signals, unless special care is made to avoid this.

A.3 STAM* overview

The 3D Signal-passing Tile Assembly Model* (3D-STAM*, or simply STAM*) was introduced in [3] as a generalization of the STAM [28, 15, 16, 23] in which (1) tiles are 3D cubes rather than 2D squares, (2) multiple tile shapes are allowed in the same system, (3) tiles are allowed to flip and rotate (similar to [8, 19]), and (4) glues are allowed to be defined as either rigid (as in the aTAM, STAM, etc.) or *flexible* (as in [13]) so that even after being bound, tiles and subassemblies are free rotate with respect to tiles and subassemblies to which they are bound by bending or twisting around a "joint" in the glue.