

# Non-cooperative algorithms in self-assembly

Pierre-Etienne Meunier\*

## Abstract

We show the first non-trivial positive algorithmic results (i.e. programs whose output is larger than their size), in a model of self-assembly that has so far resisted many attempts of formal analysis or programming: the planar non-cooperative variant of Winfree’s abstract Tile Assembly Model.

This model has been the center of several open problems and conjectures in the last fifteen years, and the first fully general results on its computational power were only proven recently (SODA 2014). These results, as well as ours, exemplify the intricate connections between computation and geometry that can occur in self-assembly.

In this model, tiles can stick to an existing assembly as soon as one of their sides matches the existing assembly. This feature contrasts with the general *cooperative* model, where it can be required that tiles match on *several* of their sides in order to bind.

In order to describe our algorithms, we also introduce a generalization of regular expressions called *baggy expression*. Finally, we compare this model to other automata-theoretic models.

## 1 Introduction

Self-assembly is the process by which unorganized atomic components coalesce into complex shapes and structures in an unsupervised way. This kind of processes is ubiquitous in nature, and in particular in the complex molecular components of life. In recent years, its study has yielded a growing number of impressive experimental realizations, ranging from regular arrays [30] to fractal structures [11, 23], smiling faces [21, 28], DNA tweezers [31], logic circuits [19, 24], neural networks [20], and molecular robots [15].

Potential future applications range from more efficient, cheaper computational units to interactions with natural biological processes, both for medical diagnosis and treatment, and a better understanding of evolution and development.

Realizing that *programming* these processes is the keystone of atomically precise molecular engineering, Winfree introduced in 1998 the abstract Tile Assembly Model [29] to program assemblies using the components built by Seeman [25] using DNA. This model is similar to Wang tilings [27], essentially augmented with a mechanism for sequential growth, and thus allowing mismatches between adjacent tiles. More precisely, in the abstract Tile Assembly Model, we consider square tiles from a finite set of *types*, with *colors* and integer *glue strengths* on each side. The assembly starts from a single “seed” tile, and proceeds by adding one tile at a time, asynchronously and nondeterministically. At each step, a tile can stick to the current assembly if the glue strengths, on its sides whose colors match the current assembly, sum up to at least a parameter of the model called the *temperature*  $\tau = 1, 2, 3 \dots$

---

\*Aix Marseille Université, CNRS, LIF UMR 7279, 13288, Marseille, France, pierre-etienne.meunier@lif.univ-mrs.fr. Supported in part by National Science Foundation Grant CCF-1219274.

In the present work, we are mostly interested in the case of temperature 1 self-assembly, also called non-cooperative self-assembly. In the abstract Tile Assembly Model, when the temperature increases, fewer assemblies are possible, allowing more control over producible assemblies: for instance, cooperative self-assembly (i.e. at temperature at least 2) is able to simulate arbitrary Turing machines [14, 22, 29], and produce arbitrary connected shapes with a number of tile types within a log factor of their Kolmogorov complexity [26]. More surprisingly, this model has even been shown intrinsically universal [8], meaning that there is a single tiling set capable of simulating arbitrary tile assembly systems, modulo rescaling, even with a single tile type [6].

Despite its apparent simplicity, the non-cooperative model is far from being well understood, and not known to be capable of general Turing computation. However, this is a fundamental and ubiquitous form of growth in nature, as many systems, from plants to mycelium to percolation processes, exhibit this kind of behavior by growing and branching tips.

In one of the first studies on self-assembly [22], Rothmund and Winfree conjectured it to be less powerful than cooperative self-assembly. The first fully general separation result, without unproven hypotheses, was only proven recently [18], in the context of intrinsic universality [7–9]. Before that, several results had shown separations between particular cases of the model [3, 10, 17], and general self-assembly.

One of the most puzzling results on this model is its capability to simulate Turing machines in the three dimensional generalization of the model [5], whereas in one dimension, it is equivalent to finite automata.

## 1.1 Main results

Here, we present the first efficient constructions in the fully general planar noncooperative model. The generally accepted definition of an “efficient program”, in this context, is a program whose output is larger than its size. Of course, a simple first result on this model shows that arbitrary shapes can be built with a number of tile types equal to the number of tiles in the shape, or (for simpler shapes) equal to the Manhattan diameter of the shape [22].

Surprisingly, our results show that there are tile assembly systems whose terminal assemblies are all larger (in Manhattan diameter) than their number of tile types. Although a number of terms have not been defined yet, we briefly introduce our two main constructions. The first construction can be proven easily by hand; we will demonstrate it first in Section 3.2, and then generalize it in Section 3.3, to get the following theorem:

**Theorem 3.4.** For all integer  $n$ , there is a tile assembly system  $\mathcal{T}_n = (T_n, \sigma_n, 1)$  such that  $|T_n| = n$ , and for all terminal assembly  $a \in \mathcal{A}_\square[\mathcal{T}_n]$ ,  $a$  is finite and of height  $2n + o(n)$ .

Intuitively, this construction works by preventing subpaths starting and ending with the same tile type to repeat completely. However, it does not address the possibility that some paths be efficient by repeating a subpath several times, before being blocked. Since these partial pumping have been a major puzzle of the field, we provide a second efficient construction allowing it.

However, its proof is significantly more complicated, and a generalized form of our construction does not seem easy. In Section 3.4, we present the computer-aided proof of its efficiency, that we have needed due to the size at which the first “savings” of tile types are seen. Computer-aided proofs are of growing importance in computer science and mathematics, as exemplified by its latest developments in complexity theory, also in the context of tile assembly [13].

However, our case here is significantly simpler, since this construction could be verified by hand, probably within a few hours:

**Theorem 3.5.** There is a tile assembly system  $\mathcal{T} = (T, \sigma, 1)$  such that  $|\text{dom}(\sigma)| = 1$ , and all terminal assemblies of  $\mathcal{T}$  contain a path  $P$  of Manhattan diameter strictly larger than  $|T| + 1$ , that is partially pumped, i.e. parts of  $P$  are consecutive repetitions of one of its subpath.

Finally, we compare this model, and several generalizations of it, to various models of automata: finite automata, tree automata, and pushdown automata. These models are explained in Section 4, and the comparison is summarized on Figure 5.

## 1.2 Key technical ideas and methods

A major challenge, when studying non-cooperative self-assembly, is to overcome the intuition given by the one-dimensional case (which is equivalent to finite automata), that any repetition of a tile type may allow to “pump” an assembly. Indeed, an easy observation shows that assemblies formed at temperature 1 are nothing more than a collection of paths growing from the seed: if a tile type is ever repeated along a path, it is tempting to try to repeat the subpath between these repetitions.

However, geometry makes things more complex. First, there are simple counter-examples to this pumping idea. Moreover, paths could first lay “blocking parts” out, and then come back and branch to check which type of blocker has been formed; this is for instance the primary mechanism used by the simulation of Turing machines in 3d shown in [5]. However, their construction “fakes cooperation” by laying a blocker out for all alternatives but one.

On the other hand, recent (unpublished) progresses tend to show that this kind of “bit reading” gadgets is not possible in two dimensions. This model thus asks a different question: can you write efficient programs without the ability to read your workspace?

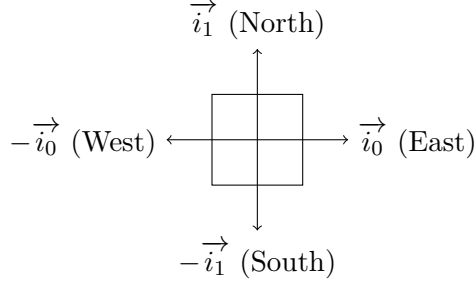
Our results show that this is possible, at least to some extent. They do so by carefully considering the fact that paths that are monotonic in one dimension are pumpable; therefore, we must build “caves”, i.e. subpaths that are non-monotonic in both dimensions. However, since these are more expensive to build than straight paths, we also need to reuse these extra tile types several times, either by making these subpaths self-blocking (in Section 3.3), and branching before the blocking, or by allowing *some* pumping (in Section 3.4) before blocking it.

These results are quite puzzling and counter-intuitive; however, they do not seem to make Turing computation possible. Therefore, a natural question is the exact power of this model, that depends strongly on geometry, and that no other “classical” model seems to capture, as shown in Section 4.

## 2 Definitions and preliminaries

We begin by defining the abstract tile assembly model, in a slightly more general framework than usually. Let  $G$  be a group with  $n$  generators  $\vec{i}_0, \vec{i}_1, \dots, \vec{i}_{n-1}$ , and arbitrary relators. We will use  $G$  to define the geometric space: for instance,  $\mathbb{Z}^2$  has two generators  $\vec{i}_0 = (1, 0)$  and  $\vec{i}_1 = (0, 1)$ , and one relation  $\vec{i}_0 \vec{i}_1 = \vec{i}_1 \vec{i}_0$ .

A *tile type* is a unit square with  $2n$  sides, each consisting of a glue *label* and a nonnegative integer *strength*. In the most common case where  $n = 2$ , we call a tile’s sides north, east, south, and west, respectively, according to the following picture:



Also, we write these directions N, E, S and W, respectively. When there is no ambiguity, we also write  $N(t)$ ,  $E(t)$ ,  $S(t)$  and  $W(t)$ , to mean the north, east, south and west glue of tile type  $t$ , respectively. Moreover, for each direction  $d$ , we write  $-d$  its opposite direction. We assume a finite set  $T$  of tile types, but an infinite supply of copies of each type. An *assembly* is a positioning of the tiles on the Cayley graph of  $G$ , that is, a partial function  $\alpha : G \dashrightarrow T$ . To simplify the notations, we will assume  $G = \mathbb{Z}^2$  throughout the paper, unless explicitly mentioned.

In this context, we say that two elements  $g_0, g_1 \in G$  are *adjacent* if  $g_1 = g_0 + \vec{i}_k$  (respectively  $g_1 = g_0 - \vec{i}_k$ ) for some generator  $\vec{i}_k$ . In this case, their *abutting side* is the  $\vec{i}_k$  side (respectively the  $-\vec{i}_k$  side) of  $g_0$ , and the  $-\vec{i}_k$  side (respectively the  $\vec{i}_k$  side) of  $g_1$ .

We say that two tiles in an assembly *interact*, or are *stably attached*, if the glue labels on their abutting side are equal, and have positive strength. An assembly  $\alpha$  induces a weighted *binding graph*  $G_\alpha = (V_\alpha, E_\alpha)$ , where  $V_\alpha = \text{dom}(\alpha)$  (the domain of  $\alpha$ ), and there is an edge  $(a, b) \in E_\alpha$  if and only if  $a$  and  $b$  interact, and this edge is weighted by the glue strength of that interaction. The assembly is said to be  $\tau$ -stable if any cut of  $G_\alpha$  has weight at least  $\tau$ .

A *tile assembly system* is a triple  $\mathcal{T} = (T, \sigma, \tau)$ , where  $T$  is a finite tile set,  $\sigma$  is called the *seed*, and  $\tau$  is the *temperature*. Throughout this paper, we will always have  $\tau = 1$ , and  $\sigma$  will always be an assembly with exactly one tile. Therefore, we can make the simplifying assumption that all glues have strength one without changing the behavior of the model.

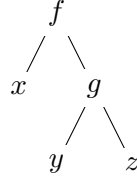
Given two  $\tau$ -stable assemblies  $\alpha$  and  $\beta$ , we say that  $\alpha$  is a *subassembly* of  $\beta$ , and write  $\alpha \sqsubseteq \beta$ , if  $\text{dom}(\alpha) \subseteq \text{dom}(\beta)$  and for all  $p \in \text{dom}(\alpha)$ ,  $\alpha(p) = \beta(p)$ . We also write  $\alpha \rightarrow_1^T \beta$  if we can get  $\beta$  from  $\alpha$  by the binding of a single tile, that is, if  $\alpha \sqsubseteq \beta$  and  $|\text{dom}(\beta) \setminus \text{dom}(\alpha)| = 1$ . We say that  $\gamma$  is *producible* from  $\alpha$ , and write  $\alpha \rightarrow^T \gamma$  if there is a (possibly empty) sequence  $\alpha = \alpha_1, \dots, \alpha_n = \gamma$  such that  $\alpha_1 \rightarrow_1^T \dots \rightarrow_1^T \alpha_n$ .

A sequence of  $k \in \mathbb{Z}^+ \cup \{\infty\}$  assemblies  $\alpha_0, \alpha_1, \dots$  over  $\mathcal{A}^T$  is a  $\mathcal{T}$ -*assembly sequence* if, for all  $1 \leq i < k$ ,  $\alpha_{i-1} \rightarrow_1^T \alpha_i$ .

The set of *productions* of a tile assembly system  $\mathcal{T} = (T, \sigma, \tau)$ , written  $\mathcal{A}[\mathcal{T}]$ , is the set of all assemblies producible from  $\sigma$ . An assembly  $\alpha$  is called *terminal* if there is no  $\beta$  such that  $\alpha \rightarrow_1^T \beta$ . The set of terminal assemblies is written  $\mathcal{A}_\square[\mathcal{T}]$ .

The *Manhattan distance*  $\|\vec{AB}\|_1$  between two points  $A = (x_A, y_A)$  and  $B = (x_B, y_B)$  is  $\|\vec{AB}\|_1 = |x_A - x_B| + |y_A - y_B|$ . The *Manhattan diameter* of a connected assembly is the maximal Manhattan distance between two points in the assembly. We write  $(u_n)_{n \in \mathbb{N}}$  to mean “the infinite sequence  $u_0, u_1, u_2, \dots$ ”.

A *regular tree grammar*  $G = (S, N, \mathcal{F}, R)$ , according to [4], is given by an *axiom*  $S$ , a set  $N$  of *nonterminal symbols*, a set  $\mathcal{F}$  of *terminal symbols*, and a set  $R$  of *production rules* of the form  $A \rightarrow \beta$  where  $A$  is a nonterminal and  $\beta$  is a tree whose nodes are labeled by elements of  $\mathcal{F} \cup N$ . Moreover, it is required that  $\mathcal{F} \cap N = \emptyset$ . In this work, we write trees as “nested function applications”: for instance,  $f(x, g(y, z))$  is the following tree:



The classical example of a regular tree grammar is the grammar of lists of integers, with one axiom *List*, non-terminals *List* and *Nat*, terminals 0, *nil*, *s()* and *cons(,)*, and the following rules:

$$\begin{aligned}
 List &\rightarrow nil \\
 List &\rightarrow cons(Nat, List) \\
 Nat &\rightarrow 0 \\
 Nat &\rightarrow s(Nat)
 \end{aligned}$$

### 3 Efficient algorithms

In this section, we show the main ideas of our efficient tileset. In order to describe them unambiguously, we use two different tools: figures showing the complete tileset and seed on the one hand, and programs written in a generalization of regular expressions called *baggin's expressions*. An implementation of these expressions using a “sublanguage” of Haskell (i.e. a monad) is available at <http://hackage.haskell.org/package/Baggins>.

Moreover, all the constructions of this paper were generated in this language, and their source code is available on the self-assembly wiki<sup>1</sup>.

#### 3.1 Baggins expressions

A program in this language is an *expr*, where *expr* is defined by the following grammar (where an *identifier* is a name):

$$\begin{aligned}
 \text{expr} &:= \text{atom} \mid \text{let} \mid \text{bind} \mid \text{from} \mid \text{expr} ; \text{expr} \\
 \text{atom} &:= \text{moveN} \mid \text{moveE} \mid \text{moveS} \mid \text{moveW} \\
 \text{let} &:= \text{let } \text{identifier} \\
 \text{bind} &:= \text{bind } [ \text{N} \mid \text{E} \mid \text{S} \mid \text{W} ] \text{ identifier} \\
 \text{from} &:= \text{from } \text{identifier}
 \end{aligned}$$

**Definition 3.1.** Let  $e$  be a baggin's expression. Let  $\beta$  the set of its identifiers. We define the unique tileset described by  $e$  by induction on  $e$ :

Let  $T_0$  be a tileset consisting of a unique tile type  $\sigma_0$ ,  $C_0 = \sigma_0$  and  $\alpha_0$  is the function defined nowhere. Then, for all  $i \in \{0, 1, \dots, |e| - 1\}$ :

- If  $e_i = \text{moveN}$ ,  $C_{i+1} = (g_N, g_E, N(C_i), g_W)$ , and  $T_{i+1} = T_i \cup \{C_{i+1}\}$ , where  $g_N, g_E, g_W$  are all new glues, not appearing on any tile of  $T_i$ . Moreover, let  $\alpha_{i+1} = \alpha$ .

<sup>1</sup> <http://self-assembly.net/wiki/index.php?title=Baggins-expressions>

- If  $e_i = \text{moveS}$ ,  $C_{i+1} = (S(C_i), g_E, g_S, g_W)$ , and  $T_{i+1} = T_i \cup \{C_{i+1}\}$ , where  $g_S, g_E, g_W$  are all new glues, not appearing on any tile of  $T_i$ . Moreover, let  $\alpha_{i+1} = \alpha$ .
- If  $e_i = \text{moveE}$ ,  $C_{i+1} = (g_N, W(C_i), g_S, g_W)$ , and  $T_{i+1} = T_i \cup \{C_{i+1}\}$ , where  $g_N, g_S, g_W$  are all new glues, not appearing on any tile of  $T_i$ . Moreover, let  $\alpha_{i+1} = \alpha$ .
- If  $e_i = \text{moveW}$ ,  $C_{i+1} = (g_N, g_E, g_S, E(C_i))$ , and  $T_{i+1} = T_i \cup \{C_{i+1}\}$ , where  $g_N, g_E, g_S$  are all new glues, not appearing on any tile of  $T_i$ . Moreover, let  $\alpha_{i+1} = \alpha$ .
- If  $e_i = \text{let } x$ , then let  $\alpha_{i+1}$  be the function of domain  $\text{dom}(\alpha_i) \cup \{x\}$ , such that for all  $y \in \text{dom}(\alpha_i) \setminus \{x\}$ ,  $\alpha_{i+1}(y) = \alpha_i(y)$ , and  $\alpha_{i+1}(x) = C_i$ .
- If  $e_i = \text{bind } d \ x$ , where  $d \in \{N, S, E, W\}$  and  $x \in \alpha_i$ , then:
  - $C_{i+1} = C_i$ ,
  - $\alpha_{i+1} = \alpha_i$ , and
  - let  $g$  be the glue on side  $d$  of  $C_i$ , and  $-g$  be the glue on side  $-d$  of  $\alpha_i(x)$ . Then  $T_{i+1}$  is  $T_i$  where all glues on sides  $d$  and  $-d$ , that are equal to  $g'$ , are replaced with  $g$ .
- If  $e_i = \text{from } x$ , where  $x \in \alpha_i$ , let  $T_{i+1} = T_i$ ,  $\alpha_{i+1} = \alpha_i$ , and  $C_i = \alpha_i(x)$ .

**Theorem 3.2.** *Definition 3.1 is “sound and complete”, i.e. any baggins expression describes exactly one tile assembly system, and any single-seeded tile assembly system can be described by a baggins expression.*

*Proof.* We prove the two properties independently:

- First remark that the construction of Definition 3.1 defines a tileset and a seed non-ambiguously.
- Now, let  $\mathcal{T} = (T, \sigma, 1)$  be a temperature 1 tile assembly system with  $|\text{dom}(\sigma)| = 1$ . Start with  $D = \{\sigma\}$ . Then, for each tile  $t \in T \setminus D$  that can bind to a tile  $t_0 \in D$  on side  $d \in \{N, S, E, W\}$  of  $t_0$ , add **from**  $t_0$  **moved** to  $D$ . Also, from any previously created tile  $t_1$  that can bind to  $t_0$ , add **from**  $t_0$  **bind**  $d$   $t_1$  to  $D$ , if this binding has not been defined before, either directly or by operation **from**  $t_1$  **bind**  $(-d)$   $t_1$  (and do nothing else).

Clearly, this baggins expression describes  $\mathcal{T}$ , by Definition 3.1.

□

In order to make the examples in the appendix shorter and more intuitive, the actual language used in our examples differs slightly from this grammar. However, all its instructions can clearly be written using baggins expression constructs.

### 3.2 A first efficient algorithm

In this section, we call a tile assembly system  $\mathcal{T} = (T, \sigma, 1)$  *efficient* if there is an integer  $r$ , such that the Manhattan diameter of all the terminal assemblies of  $\mathcal{T}$  is strictly larger than  $|T| + |\text{dom}(\sigma)|$ , and at most  $r$ .

A simple observation on paths, is that any path that is monotonic in one dimension (i.e. the sequence  $(y_{P_i})_i$  of its y-coordinates, or the sequence  $(x_{P_i})_i$  of its x-coordinates is monotonic), and repeats a tile type, is pumpable.

Therefore, the main ingredient of efficient paths is non-monotonicity: we call a *vertical cave* (respectively *horizontal cave*) a part of a path  $P$  between two indices  $i$  and  $j$ , such that (1)  $y_{P_i} = y_{P_j}$ , (2) for all  $k < i$ ,  $y_{P_k} \leq y_{P_i}$ , and (3) for all  $k \in \{i+1, i+2, \dots, j-1\}$ ,  $y_{P_k} < y_{P_i}$ .

Our first tile assembly system  $\mathcal{T}_0$  is presented completely in Appendix A, in the form of a baggins expression. We prove it now:

**Theorem 3.3.** *For all integer  $n$ , there is a tile assembly system  $\mathcal{T}_n = (T_n, \sigma_n, 1)$  such that  $|T_n| = n$ , and for all terminal assembly  $a \in \mathcal{A}_{\square}[\mathcal{T}_n]$ ,  $a$  is finite and of height  $\frac{5(n+2)}{4} - 23$ .*

*Proof.* Let  $T_0$  be the set of tiles appearing on the lower right assembly of Figure 1, and  $\sigma_0$  be the upper left assembly of that figure.

This tileset has 38 tile types, and its terminal assemblies are of height 27; it is not efficient yet. But we will now add a number of new tile types to make it efficient. First replace the following glues (zoom in on Figure 1 to see these glue numbers, or see the large version in Appendix D):

- glue 6 by  $(6, 0)$  on the north, and  $(6, n)$  on the south,
- glue 14 by  $(14, 0)$  on the north, and  $(14, n)$  on the south,
- glue 24 by  $(24, 0)$  on the north, and  $(24, n)$  on the south,
- glue 26 by  $(26, 0)$  on the north, and  $(26, n)$  on the south,

And then for all  $i \in \{6, 14, 24, 26\}$  and  $j \in \{0, 1, \dots, n-1\}$ , add a tile type to  $T$ , with south glue  $(i, j)$  and north glue  $(i, j+1)$ . In total, we have added  $4n$  tile types, but the terminal assemblies of  $T$  grow  $5n$  higher. See Figure 2 for a larger example (saving tile type).

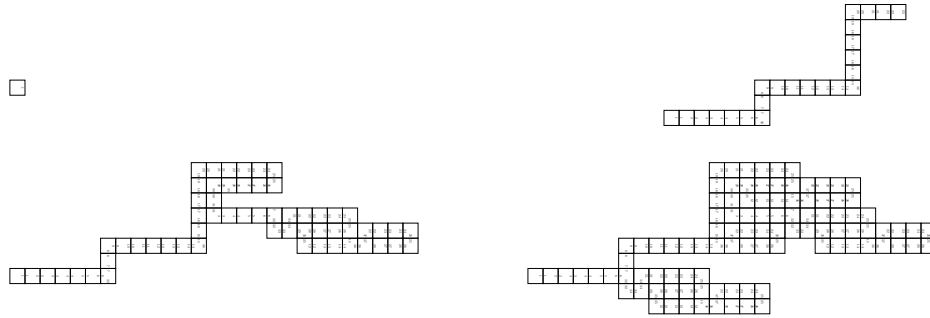


Figure 1: Four successive stages of the construction: first the seed, then the main path grows, and finally, additional branches can also grow completely, along the main path.

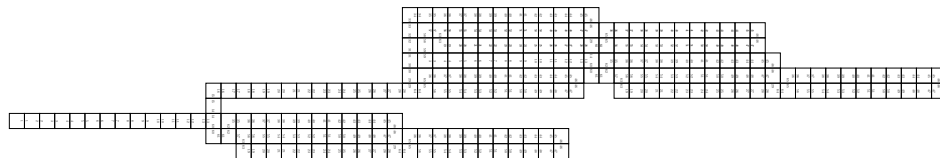


Figure 2: An efficient tile assembly system, producing an assembly of width 112 with 106 tile types. This terminal assembly grew from a seed containing only its leftmost tile.

□

### 3.3 A more general scheme

In the construction of Theorem 3.3, repetitions of a tile type are done at the expense of width of the assembly: indeed, in order to avoid collisions between repeated paths, each repetition needs to be more and more narrow. Generalizing this remark yields the following Theorem:

**Theorem 3.4.** *For all integer  $n$ , there is a tile assembly system  $\mathcal{T}_n = (T_n, \sigma_n, 1)$  such that  $|T_n| = n$ , and for all terminal assembly  $a \in \mathcal{A}_\square[\mathcal{T}_n]$ ,  $a$  is finite and of height  $2n + o(n)$ .*

*Proof.* The idea is to repeat the construction of Theorem 3.3 more than a constant number of times. A single cave, of height  $h$  (see Figure 3), will be reused  $N$  times, and at each iteration  $i \in \{0, 1, \dots, N\}$ , grow to height  $2h - i$ .

To do this, we use a sequence of assemblies as shown on Figure 3, with different widths  $(w_n)_n$ . The precise definition of this construction is given by the Haskell program in Appendix B, but the general idea is: grow some construction starting with tile type  $t$ , then use some modification of the initial cave as a blocker, and then reuse  $t$ .

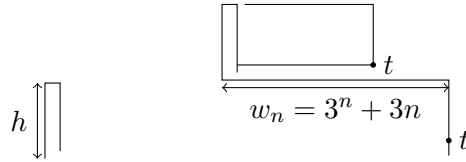


Figure 3: The repeated part is shown on the left-hand side. The drawing on the right-hand side is a scheme of one step of the construction.

Then, we stack these parts on top of each other: on the Figure 4, the next assembly, drawn in dashed line, is of width  $w_{n-1} = 3^{n-1} + 3(n-1)$ . In order to avoid making a pumpable path, we do not grow the full initial cave each time, but a smaller and smaller suffix of it at each iteration.

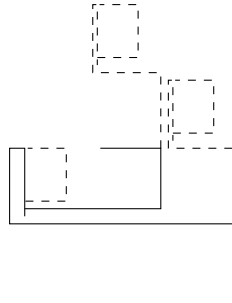


Figure 4: Two successive iterations.

Because of this choice of widths, successive assemblies cannot collide with each other, and different repetitions of the same assembly cannot collide with each other either.

Let  $h$  be the height of the initial cave. For all integer  $n$ , the  $n^{th}$  repetition requires  $w_n + 2w_{n-1} \leq 2w_n$  new tiles horizontally,  $h - n$  tiles vertically, and grows to a height of  $2(h - n)$ . If we decide to repeat the construction  $N = \log h$  times, we need  $|T| = 2 \sum_{i=1}^N w_n + Nh + O(N^2)$  tile types, i.e.  $h \log h + O(h)$  tile types.

Moreover, in this case, all terminal assemblies will have height  $2h \log h + O(N^2)$ , which is  $2|T| + o(|T|)$ .



The baggins expression for the exact construction is in Appendix B.

□

### 3.4 Partially pumpable paths

The constructions of Sections 3.2 and 3.3 are efficient by repeating smaller and smaller parts of an assembly, while ensuring that the assembly does not become pumpable. The other way of building efficient paths is by letting them become pumpable for some time, after building structures that block these repetitions. However, blocking these parts is provably expensive, and the same kind of repeated blocking structure, similar to those of Sections 3.2 and 3.3, must be used to “save” tile types. However, this construction is intended as a proof that allowing some pumping still does not forbid the existence of efficient tilesets.

**Theorem 3.5.** *There is a tile assembly system  $\mathcal{T} = (T, \sigma, 1)$  such that  $|\text{dom}(\sigma)| = 1$ , and all terminal assemblies of  $\mathcal{T}$  contain a path  $P$  of Manhattan diameter strictly larger than  $|T| + 1$ , that is partially pumped, i.e. parts of  $P$  are consecutive repetitions of one of its subpath.*

*Proof.* The smallest efficient tile assembly system that we found with a seed of size 1, has 4825 tile types, and all its terminal assemblies are of Manhattan radius 4845.

To show this, we use a computer-aided proof: more specifically, we simulate the assembly of the tile assembly system described by the baggins expression in Appendix C, yielding the assembly of Figure 6 (also in Appendix C).

Again, the full Haskell program, generating a (quite large) pdf file with the construction, can be found on the self-assembly wiki<sup>2</sup>. □

## 4 Comparisons with other models

The constructions of Section 3 show the intricate connections between geometry and the computational power of temperature 1 self-assembly, raising the question of the exact characterization of the model, from the point of view of classical computational models. In this section, we show that we are far from understanding these relations, and begin a broader exploration of the influence of geometry. In Wang tilings, geometries that have been considered previously include the hyperbolic plane [12, 16] and Cayley graphs of Baumslag-Solitar groups [1, 2].

From the self-assembly side, the models and underlying graphs that we considered are the following:

- Temperature 1 tile assembly, on  $\mathbb{Z}^2$ .
- Temperature 1 tile assembly, on the Cayley graph of Baumslag-Solitar groups.
- Temperature 1 tile assembly, on the hyperbolic plane.

From the “classical” side, the computational models that we considered are the following:

- Finite automata
- Regular tree automata
- Pushdown automata

---

<sup>2</sup> <http://self-assembly.net/wiki/index.php?title=Baggins-expressions>

- Turing machines

The results shown on Figure 5 are proven in Appendix E.

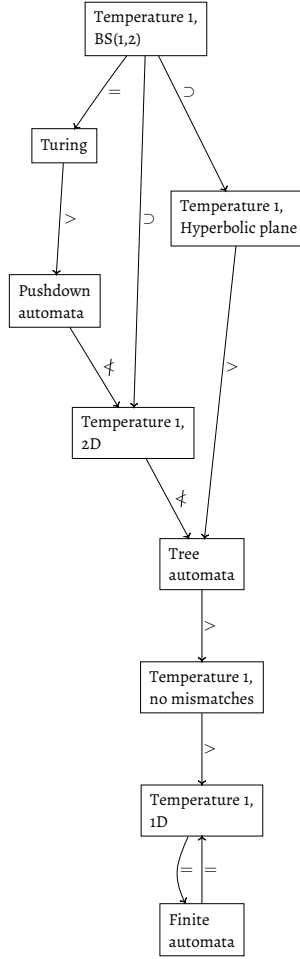


Figure 5: Summary of the comparisons of Section E. On this graph, an arrow from  $A$  to  $B$ , labeled with relation  $\mathcal{R}$  means  $A\mathcal{R}B$ .

## 5 Open problems and discussion

Despite our efficient constructions, planar temperature 1 tile assembly model does not seem capable of Turing computation. Finding the limits of these constructions would give us a greater understanding of these processes, ubiquitous in natural systems:

**Open Problem 1.** What is the largest integer  $s$ , such that all the terminal assemblies of a tile assembly system with  $n$  tiles and a single-tile seed, are of size  $s$ ?

Another question, left open by Section 4, is the exact characterization of this model, in terms of classical models.

## A A first efficient algorithm

```
programme::Int→Program ()
programme n=do
  seed 7 0
  movey 3
  a←currentTile
  movey n
  a1←currentTile
  movex (-2)
  movey 3
  b←currentTile
  movey 1
  c←currentTile
  movey (n-1)
  b1←currentTile
  movex (-5)
  gr←currentTile
  movey 2
  gr1←currentTile
  movey 1
  gr2←currentTile
  movey (n-1)
  movex 1
  movey (-n-1)
  bot←currentTile
  movex 2
  bind N a

  rewindTo a1
  movex 1
  movey 1
  movex (-1)
  bind N gr1

  rewindTo bot
  rewindBy 1
  movex 1
  bind N c
  rewindTo b1
  movey 1
  movex (-1)
  bind N gr2
```

## B A more general scheme

```
programme::Int→Int→Program ()
programme n h=do
  seed (3^n+n) 0
  -- A first occurrence of the construction creates the cave.
  movey 2
  a←currentTile
  movey (h-2)
```

```

c←currentTile
movex (-3^(n-1)-n)
-- Start of the cave
b←currentTile
movey h
movex 1 -- Top of the cave
movey (-h+1)
d←currentTile -- Bottom of the cave
-- Now, move to the right, and repeat tile a
movex (2*3^(n-2))
bind N a
-- Now, iterate n times. To avoid making the path pumpable, we need to reduce
-- the height (parameter hh) each time.
let prog n hh b0 d0=
  if n≤0 then
    return ()
  else do
    movey 1
    an←currentTile
    movey (hh)
    cn←currentTile
    movex (-3^(n)+n)
    bind N b0
    rewindTo d0
    movex (2*3^(n-1)-n)
    bind N an
    b1←nextTile b0
    d1←prevTile d0
    rewindTo cn
    prog (n-1) (hh-1) b1 d1
-- Go back to tile c, and start iterating.
rewindTo c
b0←nextTile b
b1←nextTile b0
d0←prevTile d
prog (n-2) (h-3) b1 d0

```

## C A partially pumpable path

This program is slightly more complex than those of Sections A and B. We tried to stick to basic parts of Haskell syntax; the main things that need to be explained are the following:

- the “let” syntax we use here is the Haskell way of defining variables, and is not related to the let construct of baggins expressions.
- for reasons of efficiency, we need a new instruction called `discreteVect`. It is built using `movex` and `movey` instructions, combined in an efficient way.
- `quot` means “quotient”.

```

programme::Program ()
programme=do
  seed 0 0

```

```

movey 1
a0←currentTile
-- First step: grow the part that will be repeated.
--
-- Since we want to grow upwards, and then follow it downwards closely,
-- we need precise control over its shape.
repete 15
  (do
    repete 20 (do { movey 2;movex 1 })
    movex 1)
a←nextTile a0
-- Now, lay a "blocker" out, for the partially pumped paths to stop.
movex 1
movey (-1)
movex (-2)
-- Then go down.
repete 15
  (do
    repete 20 (do { movey (-2);movex (-1) })
    movex (-1))
-- Now, build the bottom of the construction.
rewindBy 6
c←currentTile
eraseAfter c
movey (-1)

let x2=40
    tot=21*15-1-x2
    x0=15
    x1=(tot-x0)`quot`3-15

-- Record the three different starting tiles of exit paths.
movex x0
start0←currentTile
movex x1
start1←currentTile
movex (tot-x1-x0)
start2←currentTile
-- Now, from each starting tile, grow a partially pumpable path, that will be
-- blocked on its way up.

-- First exit path.
rewindTo start2
pump (do
  setColor blue
  discreteVect 16 (16*15-3))

-- Here, the transition to the next repetition is simple: we just move to the
-- right by 120 columns, lay a blocker out, so that the repeated part (from
-- tile a) cannot be repeated completely.
rewindBy 3
movex 120
movey 2
movex (-1)

```

```

movey (-1)
movex (-x1-x0-5)
bind N a -- Finally, start the repeated part again.

-- The second exit path is more complicated, since we do not want it to
-- collide with the first pumped path. Moreover, the discreteVect function is
-- used to build the most efficient vector (in terms of number of tile types)
-- with the given coordinates.
rewindTo start1
pump
  (do
    setColor red
    let distx=tot-x0-x1+x2+1
        disty=15*40
        discreteVect (distx`quot`2) (disty`quot`2))
rewindBy 51
movex 10
repete 7
  (do
    repete 20 (do { movey 2; movex 1 })
    movex 1)

movex (x0+4)
movey 2
movex (-1)
movey (-1)
movex (-x0-4)
bind N a

-- The "final" exit path is similar, but simpler: we closely follow the
-- repeated part.
rewindTo start0
movey 1
movex 1
pump
  (do
    setColor green
    repete 149 (do { movey 2; movex 1 })))

movex 2
repete 2
  (do
    repete 148 (do { movey 2; movex 1 })
    movex 2)

movex 10
movey 2
movex (-1)
movey (-1)
movex (-2)
bind N a

```

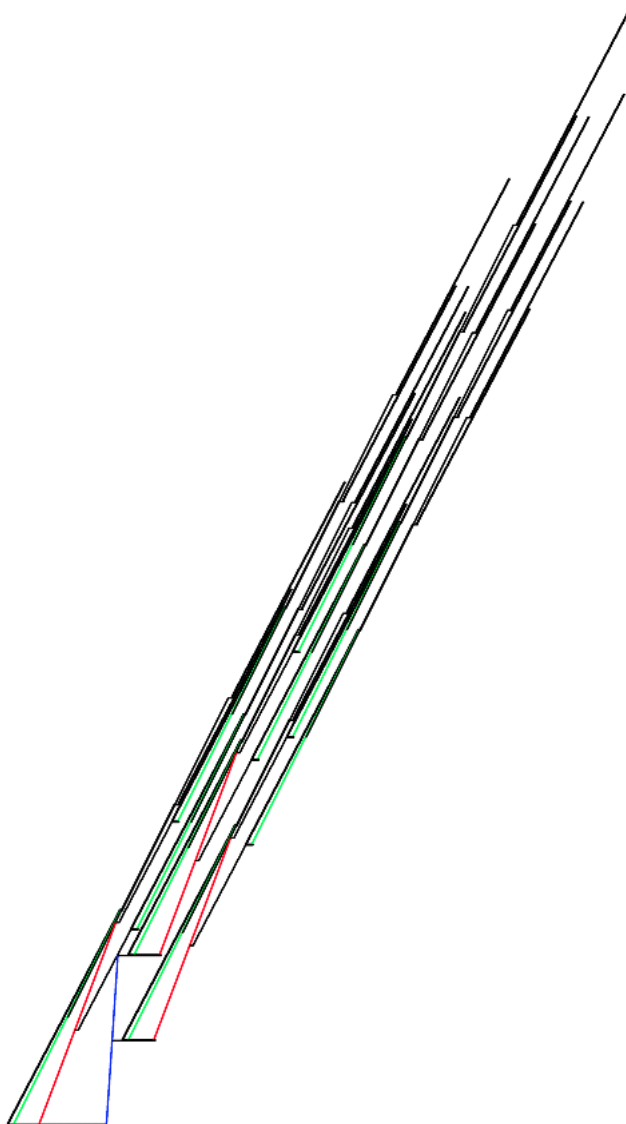


Figure 6: A partially pumpable efficient path. The three successive partially pumped parts are colored in blue, red and green, successively. This image is rasterized for size reasons, please run the program above for a vector version.

D A printable version of Figure 1

[illegible]



## E Comparison with other models

In order to compare various settings of non-cooperative self-assembly with classical machines from automata theory, we first introduce a notion of language for a tile assembly system:

**Definition E.1.** Let  $\mathcal{T} = (T, \sigma, 1)$  be a temperature 1 tile assembly system where  $\sigma$  is single-tile seed assembly. We call  $\mathcal{L}(\mathcal{T})$ , the *language* of  $\mathcal{T}$ , the tree language recognized by the following tree grammar:

- For each tile  $t \in T$ , with glues  $t_N$  on the north,  $t_E$  on the east,  $t_S$  on the south, and  $t_W$  on the west,  $\mathcal{A}$  has the four following nonterminals:

$$\begin{aligned} N_{t_N} &\rightarrow N(E_{t_E}, S_{t_S}, W_{t_W}) \\ E_{t_E} &\rightarrow E(S_{t_S}, W_{t_W}, N_{t_N}) \\ S_{t_S} &\rightarrow S(W_{t_W}, N_{t_N}, E_{t_E}) \\ W_{t_W} &\rightarrow W(N_{t_N}, E_{t_E}, S_{t_S}) \end{aligned}$$

- Moreover, for each glue  $g$  appearing on the north (respectively south, west and east side) of some tile of  $T$ , add a terminal symbol  $n_g$  (respectively  $s_g$ ,  $w_g$ ,  $e_g$ ) to the grammar, and the following rules:

$$\begin{aligned} N_g &\rightarrow n_g \\ E_g &\rightarrow e_g \\ S_g &\rightarrow s_g \\ W_g &\rightarrow w_g \end{aligned}$$

- Finally, add a nonterminal symbol  $S$ , and the following rule:

$$S \rightarrow \Sigma(N_{\sigma_N}, E_{\sigma_E}, S_{\sigma_S}, W_{\sigma_W})$$

Where  $\sigma_N$ ,  $\sigma_E$ ,  $\sigma_S$  and  $\sigma_W$  are the north, east, south and west glues of the unique tile of  $\sigma$ , respectively.

**Definition E.2.** Let  $\mathcal{T} = (T, \sigma, 1)$  be a temperature 1 tile assembly system. A term  $t$  of  $\mathcal{L}(\mathcal{T})$  *describes* the following assembly sequence:

- From  $\Sigma(N_{\sigma_N}, E_{\sigma_E}, S_{\sigma_S}, W_{\sigma_W})$ , concatenate the four assembly sequences obtained from  $N_{\sigma_N}$ ,  $E_{\sigma_E}$ ,  $S_{\sigma_S}$ ,  $W_{\sigma_W}$ , successively.
- Let  $\alpha(x, y, n, N(E_{t_E}, S_{t_S}, W_{t_W}))$  be concatenation of the following sequences:
  - the assembly of the unique tile type  $t \in T$  with north glue  $n$ , east glue  $t_E$ , south glue  $t_S$  and west glue  $t_W$ , at position  $(x, y)$ .
  - assembly sequence  $\alpha(x + 1, y, t_E, E_{t_E})$ .
  - assembly sequence  $\alpha(x - 1, y, t_W, W_{t_W})$ .
  - assembly sequence  $\alpha(x, y - 1, t_S, S_{t_S})$ .

- Similarly for  $\alpha(x, y, e, E(S_{t_S}, W_{t_W}, N_{t_N}))$ ,  $\alpha(x, y, s, S(W_{t_W}, N_{t_N}, E_{t_E}))$ , and  $\alpha(x, y, w, W(N_{t_N}, E_{t_E}, S_{t_S}))$ .
- For terminals  $t$  of the form  $n_g, s_g, e_g$  or  $w_g$ , let  $\alpha(x, y, g, t)$  be the empty assembly sequence.

By extension, if this assembly sequence results in a producible assembly  $a \in \mathcal{A}[\mathcal{T}]$ , we say that  $t$  describes  $a$ . Moreover, if all the terms of some tree language  $L$  describe a producible assembly of  $\mathcal{T}$ , and all producible assemblies of  $\mathcal{T}$  are described by some term  $t \in L$ , we say that  $L$  describes  $\mathcal{A}[\mathcal{T}]$ .

When all the nodes of terms of  $\mathcal{L}(\mathcal{T})$  have at most one nonterminal child, this tree language is also a word language, over alphabet  $T$ .

**Proposition 1.** *Let  $A$  be a non-deterministic finite automaton on alphabet  $S$ . There is a (one-dimensional) tile assembly system  $\mathcal{T}_A = (T_A, \sigma_A, 1)$  such that  $\mathcal{L}(A)$  describes  $\mathcal{A}_\square[\mathcal{T}_A]$ .*

*Proof.* Let  $A = (Q, \Sigma, \Delta, q_0, F)$  be any non-deterministic finite automaton, with  $Q$  its set of states,  $\Sigma$  its alphabet,  $\Delta \in Q \times \Sigma \times Q$  its transition relation,  $q_0$  its start state and  $F$  its set of final states.

We build an “equivalent” temperature 1 tile assembly system  $\mathcal{T}_A = (T_A, \sigma_A, 1)$ , where  $T_A$  is a tileset with glue colors from  $Q$ , by letting:

- $t_\sigma$  be a tile with exactly one non-zero strength glue, on its east side, with color  $q_0$ .
- for each  $(q, s, q') \in \Delta$ ,  $\delta_{(q,s,q')}$  be a tile with color  $q$  on its west side,  $q'$  on its east side, and  $s$  on its north side.
- for each  $q \in F$ ,  $f_q$  be a tile with color  $q$  on its east side, and no other non-zero strength glue.

Then, let  $T_a = \{t_\sigma\} \cup \{\delta_{(q,s,q')} | (q, s, q') \in \Delta\} \cup \{f_q | q \in F\}$ , and  $\sigma_A$  be an assembly with exactly one tile of type  $t_\sigma$ , at position  $(0, 0)$ .

Clearly, the language  $\mathcal{L}(A)$  recognized by  $A$  describes the terminal assemblies of  $\mathcal{T}_A = (T_A, \sigma_A, 1)$ .  $\square$

**Proposition 2.** *For any temperature 1 tile assembly system  $\mathcal{T} = (T, \sigma, 1)$  without mismatches, and such that  $\sigma$  is a connected assembly, there is a nondeterministic top-down tree automaton whose language describes  $\mathcal{A}[\mathcal{T}]$ .*

*Proof.* Clearly, since there are no mismatches in the productions of  $\mathcal{T}$ , every assembly described by  $\mathcal{L}(\mathcal{T})$  is producible by  $\mathcal{T}$ . The other direction (producible assemblies of  $\mathcal{T}$  are described by  $\mathcal{L}(\mathcal{T})$ ) is immediate.  $\square$

**Proposition 3.** *There is a temperature 1 tile assembly system  $\mathcal{T}$  such that  $\mathcal{L}(\mathcal{T})$  describes assembly sequences not producible by  $\mathcal{T}$ .*

*Proof.* Let  $T$  be the following tileset:

$$T = \left\{ t_0 = \begin{bmatrix} a \\ \phantom{a} \end{bmatrix}, t_1 = \begin{bmatrix} a \\ a \end{bmatrix}, t_2 = \begin{bmatrix} \phantom{a} \\ a \end{bmatrix}, t_3 = \begin{bmatrix} a & a \end{bmatrix}, t_4 = \begin{bmatrix} a & b \end{bmatrix}, t_5 = \begin{bmatrix} b \\ b \end{bmatrix}, t_6 = \begin{bmatrix} c & b \end{bmatrix}, t_7 = \begin{bmatrix} c & c \end{bmatrix} \right\}$$

Let  $\sigma$  be the assembly with a single tile of type  $t_0$ .

We claim that for  $\mathcal{T} = (T, \sigma, 1)$ ,  $\mathcal{L}(\mathcal{T})$  describes assembly sequences not representing any assembly. First, since all the tiles of  $T$  can attach to at most two tiles, we can completely describe

assembly sequences as words on  $T$ . Let  $L$  be the language of all assembly sequences ( $L$  is therefore a word language on alphabet  $T$ ).

Since  $\mathcal{L}(\mathcal{T})$  is a regular tree language,  $L$  is a regular language, and is therefore recognized by a deterministic finite automaton  $A$ . Let  $n$  be the number of states of  $A$ , and let  $u = t_0 t_1^n t_2 t_4 t_5^{n+1} t_6 t_7^{10}$ . Moreover, for  $i \in \{0, 1, \dots, |u| - 1\}$ , let  $a_i$  be the state in which  $A$  is just before letter  $u_i$ . Since there are  $n + 1$  occurrences of  $t_5$  in  $u$ , at least two distinct indices  $i$  and  $j$ , in subword  $t_5^{n+1}$  of  $u$ , are such that  $a_i = a_j$ .

This means that the following word, which does not described any production of  $\mathcal{T}$ , is recognized:  $t_0 t_1^n t_2 t_4 t_5^{n+1-b+a} t_6 t_7^{10}$ . □

**Proposition 4.** *There is a temperature 1 tile assembly system  $\mathcal{T} = (T, \sigma, 1)$  such that  $\mathcal{L}(\mathcal{T})$  is a non-context-free word language on alphabet  $T$ .*

*Proof.* Let  $T$  be the following tileset:

$$T = \left\{ t_0 = \begin{bmatrix} a_0 \\ \phantom{a_0} \end{bmatrix}, t_1 = \begin{bmatrix} a_1 \\ a_0 \end{bmatrix}, t_2 = \begin{bmatrix} a_1 & b \end{bmatrix}, t_3 = \begin{bmatrix} b & b \end{bmatrix}, t_4 = \begin{bmatrix} b & c_2 \\ c_2 \end{bmatrix}, t_5 = \begin{bmatrix} c_2 \\ c_1 \end{bmatrix}, \right. \\ \left. t_6 = \begin{bmatrix} c_1 \\ d \end{bmatrix}, t_7 = \begin{bmatrix} d & d \end{bmatrix}, t_8 = \begin{bmatrix} e \\ d \end{bmatrix}, t_9 = \begin{bmatrix} e & f \\ e \end{bmatrix}, t_{10} = \begin{bmatrix} f & f \end{bmatrix} \right\}$$

Since all tiles of  $T$  have exactly two sides of non-zero strength, the tree language  $\mathcal{L}(\mathcal{T})$  is actually also a word language, on alphabet  $T$ . However, the language  $L$  of the productions of  $T$  is the union of the language  $M$  describing the terminal assemblies of  $\mathcal{T}$ , with all the prefixes of these assemblies. Formally,  $M$  is the following language:

$$M = \{t_0 t_1 t_2 t_3^a t_4 t_5 t_6 t_7^b t_8 t_9 t_{10}^c \mid a > b \geq c\} \cup \{t_0 t_1 t_2 t_3^a t_4 t_5 t_6 t_7^a \mid a \in \mathbb{N}\}$$

Moreover, by the pumping Lemma on pushdown automata, this means if  $L$  were context-free, then it would also contain words of the form  $t_0 t_1 t_2 t_3^a t_4 t_5 t_6 t_7^b t_8 t_9 t_{10}^c$  in which either  $c > b$  or  $b \geq a$ , which is not the case. Indeed, for all  $a$ ,  $M$  contains the following word:

$$t_0 t_1 t_2 t_3^{a+1} t_4 t_5 t_6 t_7^a t_8 t_9 t_{10}^a$$

Therefore, the pumping lemma states that  $L$  were context-free, it would also contain:

- Either  $t_0 t_1 t_2 t_3^{a+1-b} t_4 t_5 t_6 t_7^{a-b} t_8 t_9 t_{10}^a$  for some  $b < a$ . However, this word is not in  $L$ .
  - Or  $t_0 t_1 t_2 t_3^{a+1} t_4 t_5 t_6 t_7^{a+b} t_8 t_9 t_{10}^{a+b}$  for some  $b > 0$ , which is also not in  $L$ .
  - Or  $t_0 t_1 t_2 t_3^{a+1+b} t_4 t_5 t_6 t_7^a t_8 t_9 t_{10}^{a+b}$  for some  $b > 0$ , which is also not in  $L$ .
- 

**Definition E.3.** A Baumslag-Solitar group of integer parameters  $m$  and  $n$  is a group given by the following presentation (with two generators  $a$  and  $b$ , and one relation):

$$B(m, n) = \langle a, b \mid ba^m = a^n b \rangle$$

**Proposition 5.** *For any Turing machine  $M$  and all input  $x \in \mathbb{N}$  for  $M$ , there is a tile assembly system  $\mathcal{T}_{M,x} = (T_M, \sigma_{M,x}, 1)$  on Baumslag-Solitar group  $B(1, 2)$ , and a tile  $t \in T_M$ , such that:*

- $\sigma_{M,x}$  is recursive

- all terminal assemblies of  $\mathcal{T}_{M,x}$  contain  $t$  if and only if  $M$  accepts  $x$ .

*Proof.* This is a straightforward adaptation of the 3D construction of Cook, Fu and Schweller [5], simulating zig-zag systems (and thus Turing machines).

The geometric intuition is that  $B(1, 2)$  is a “tree of half-planes” (see Figure 7). In the construction, we will most of the time stay in the “initial” plane, i.e. the leftmost branch of the tree, and avoid planarity by taking another branch temporarily.

Now, contrarily to the grid graph of  $\mathbb{Z}^3$ , there is no edge in the Cayley graph of  $BS(1, 2)$  between these “half planes”.

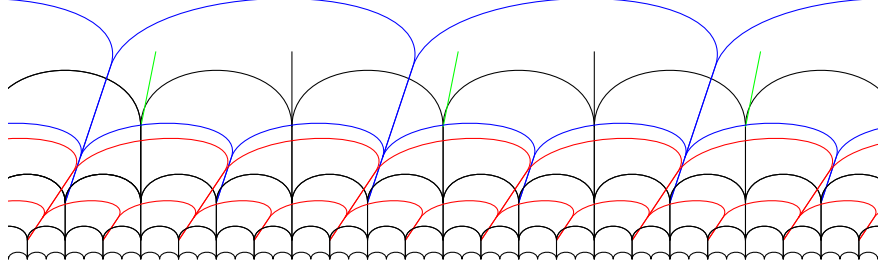


Figure 7: Some points and relations of  $BS(1, 2)$ . Different “half-planes” are in different colors.

However, their bit selection gadget can be adapted to  $BS(1, 2)$ , in the way depicted on Figure 8: the red and green paths encode a zero or a one. In order to read it, the orange path forks into two branches, and only one is allowed to pass through the encoding (the other one collides against a part of the encoded bit).

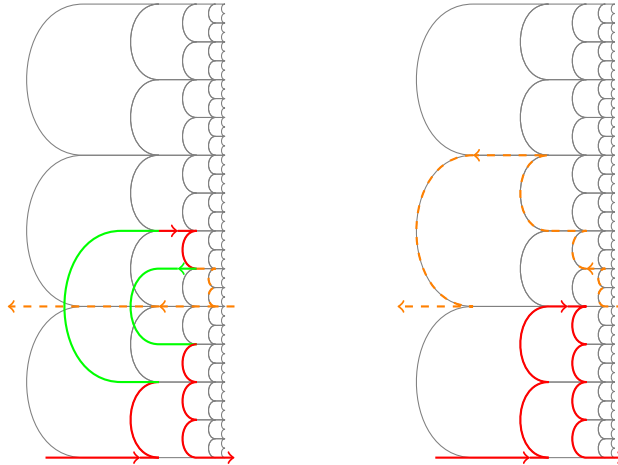


Figure 8: Adapting the bit selection gadget of [5] to  $BS(1, 2)$ . In this figure, the red/green paths grow first, and encode a 0 on the left assembly, and a 1 on the right one. The parts of the initial paths that are on the first “plane” are in red, other parts are in green. The orange (dashed) paths are paths from the next row, that read this encoding.

□

**Proposition 6.** *Let  $L$  be a regular tree language of degree at most  $d$ . There is a tile assembly system  $(T, \sigma, 1)$  in the hyperbolic plane, where  $|\text{dom}(\sigma)| = 1$ , and such that  $\mathcal{A}[T]$  is described by  $L$ .*

*Proof.* The hyperbolic plane is a tree of degree  $k$ , along with edges between consecutive vertices of the same level, and an edge between the first and last vertices of each level (see [16] for more details).

Therefore, simulating a tree automaton of degree  $k$  is straightforward, and Definition E.1 allows us to conclude.  $\square$

## References

- [1] Nathalie Aubrun and Jarkko Kari. Tiling problems on baumslag-solitar groups. In *Proceedings of the 8th International Conference on Machines, Computations, and Universality*, MCU 2013, 2013.
- [2] A. Ballier and M. Stein. The domino problem on groups of polynomial growth. *ArXiv e-prints*, November 2013.
- [3] Harish Chandran, Nikhil Gopalkrishnan, and John Reif. Tile complexity of approximate squares. *Algorithmica*, 66(1):1–17, 2013.
- [4] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. Available on: <http://www.grappa.univ-lille3.fr/tata>, 2007. release October, 12th 2007.
- [5] Matthew Cook, Yunhui Fu, and Robert T. Schweller. Temperature 1 self-assembly: deterministic assembly in 3D and probabilistic assembly in 2D. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 570–589, 2011. Arxiv preprint: [arXiv:0912.0027](https://arxiv.org/abs/0912.0027).
- [6] Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Matthew J. Patitz, Robert T. Schweller, Andrew Winslow, and Damien Woods. One tile to rule them all: Simulating any tile assembly system with a single universal tile. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP (1)*, volume 8572 of *Lecture Notes in Computer Science*, pages 368–379. Springer, 2014. Arxiv preprint: [arXiv:1212.4756](https://arxiv.org/abs/1212.4756).
- [7] Erik D. Demaine, Matthew J. Patitz, Trent A. Rogers, Robert T. Schweller, Scott M. Summers, and Damien Woods. The two-handed tile assembly model is not intrinsically universal. In *ICALP: 40th International Colloquium on Automata, Languages and Programming*, volume 7965 of *LNCS*, pages 400–412, Riga, Latvia, July 2013. Springer. Arxiv preprint: [arXiv:1306.6710](https://arxiv.org/abs/1306.6710).
- [8] 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*, pages 439–446, October 2012. Arxiv preprint: [arXiv:1111.3097](https://arxiv.org/abs/1111.3097).
- [9] David Doty, Jack H. Lutz, Matthew J. Patitz, Scott M. Summers, and Damien Woods. Intrinsic universality in self-assembly. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science*, pages 275–286, 2009. Arxiv preprint: [arXiv:1001.0208](https://arxiv.org/abs/1001.0208).
- [10] David Doty, Matthew J. Patitz, and Scott M. Summers. Limitations of self-assembly at temperature 1. *Theoretical Computer Science*, 412(1–2):145–158, 2011. Arxiv preprint: [arXiv:0906.3251](https://arxiv.org/abs/0906.3251).

- [11] Kenichi Fujibayashi, Rizal Hariadi, Sung Ha Park, Erik Winfree, and Satoshi Murata. Toward reliable algorithmic self-assembly of DNA tiles: A fixed-width cellular automaton pattern. *Nano Letters*, 8(7):1791–1797, 2007.
- [12] Jarkko Kari. The tiling problem revisited (extended abstract). In *Proceedings of the 5th International Conference on Machines, Computations, and Universality*, MCU 2007, pages 72–79, 2007.
- [13] Lila Kari, Steffen Kopecki, Pierre-Étienne Meunier, Matthew Patitz, and Shinnosuke Seki. Binary pattern tileset synthesis is np-hard, 2014. Arxiv preprint: [arXiv:1404.0967](#).
- [14] 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.
- [15] Kyle Lund, Anthony T. Manzo, Nadine Dabby, Nicole Micholotti, Alexander Johnson-Buck, Jeanetter Nangreave, Steven Taylor, Renjun Pei, Milan N. Stojanovic, Nils G. Walter, Erik Winfree, and Hao Yan. Molecular robots guided by prescriptive landscapes. *Nature*, 465:206–210, 2010.
- [16] Maurice Margenstern. About the domino problem in the hyperbolic plane, a new solution. ArXiv preprint: [cs/0701096](#).
- [17] Ján Maňuch, Ladislav Stacho, and Christine Stoll. Two lower bounds for self-assemblies at temperature 1. *Journal of Computational Biology*, 17(6):841–852, 2010.
- [18] Pierre-Étienne Meunier, Matthew J. Patitz, Scott M. Summers, Guillaume Theyssier, Andrew Winslow, and Damien Woods. Intrinsic universality in tile self-assembly requires cooperation. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 752–771, 2014. Arxiv preprint: [arXiv:1304.1679](#).
- [19] Lulu Qian and Erik Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332(6034):1196, 2011.
- [20] Lulu Qian, Erik Winfree, and Jehoshua Bruck. Neural network computation with DNA strand displacement cascades. *Nature*, 475(7356):368–372, 2011.
- [21] Paul W. K. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, March 2006.
- [22] 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.
- [23] Paul W.K. Rothemund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology*, 2(12):2041–2053, 2004.
- [24] Georg Seelig, David Soloveichik, David Yu Zhang, and Erik Winfree. Enzyme-free nucleic acid logic circuits. *Science*, 314(5805):1585–1588, 2006.
- [25] Nadrian C. Seeman. Nucleic-acid junctions and lattices. *Journal of Theoretical Biology*, 99:237–247, 1982.

- [26] David Soloveichik and Erik Winfree. Complexity of self-assembled shapes. *SIAM Journal on Computing*, 36(6):1544–1569, 2007. Arxiv preprint: [arXiv:cs/0412096](https://arxiv.org/abs/cs/0412096).
- [27] Hao Wang. Proving theorems by pattern recognition – II. *The Bell System Technical Journal*, XL(1):1–41, 1961.
- [28] Bryan Wei, Mingjie Dai, and Peng Yin. Complex shapes self-assembled from single-stranded DNA tiles. *Nature*, 485(7400):623–626, 2012.
- [29] Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.
- [30] Erik Winfree, Furong Liu, Lisa A. Wenzler, and Nadrian C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394(6693):539–44, 1998.
- [31] Bernard Yurke, Andrew J Turberfield, Allen P Mills, Friedrich C Simmel, and Jennifer L Neumann. A DNA-fuelled molecular machine made of DNA. *Nature*, 406(6796):605–608, 2000.