Available online at www.sciencedirect.com

ScienceDirect

Theoretical Computer Science 378 (2007) 17–31

Theoretical Computer Science

www.elsevier.com/locate/tcs

Arithmetic computation in the tile assembly model: Addition and multiplication

Yuriy Brun[*]

Department of Computer Science, University of Southern California, Los Angeles, CA 90089-2910, United States

Received 12 May 2006; received in revised form 26 October 2006; accepted 27 October 2006

Communicated by A. Condon

Abstract

Formalized study of self-assembly has led to the definition of the tile assembly model [Erik Winfree, Algorithmic self-assembly of DNA, Ph.D. Thesis, Caltech, Pasadena, CA, June 1998; Paul Rothemund, Erik Winfree, The program-size complexity of self-assembled squares, in: ACM Symposium on Theory of Computing, STOC02, Montreal, Quebec, Canada, 2001, pp. 459–468]. Research has identified two issues at the heart of self-assembling systems: the number of steps it takes for an assembly to complete, assuming maximum parallelism, and the minimal number of tiles necessary to assemble a shape. In this paper, I define the notion of a tile assembly system that computes a function, and tackle these issues for systems that compute the sum and product of two numbers. I demonstrate constructions of such systems with optimal $\Theta(1)$ distinct tile types and prove the assembly time is linear in the size of the input.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Self-assembly; Adder; Multiplier; Tile assembly model; Crystal growth; Molecular computation

1. Introduction

Self-assembly is a crucial process by which systems form in nature on all scales. Atoms self-assemble to form molecules, molecules to form compounds, and stars and planets to form galaxies. One aspect of self-assembly is crystal growth: molecules self-assembling to form crystals. Crystal growth is an interesting area of research for computer scientists because it has been shown that, in theory, under careful control crystals can compute [23]. The field of DNA computation demonstrates that DNA can be used to compute [3], solving NP-complete problems such as the satisfiability problem [10,7]. Winfree showed that DNA computation is Turing-universal [22]. While DNA computation suffers from relatively high error rates, the study of self-assembly shows how to utilize redundancy for error correction [21,9,12,18]. Researchers have used DNA to assemble crystals with patterns of binary counters [11] and Sierpinski triangles [17].

Two important questions about self-assembling systems that create shapes or compute functions are: “What is a minimal tile set that can accomplish this goal?” and “What is the minimum assembly time for this system?” Here, I will ask these questions for systems that compute the sum and product of two numbers.

[*] Tel.: +1 3023545174.
E-mail address: ybrun@usc.edu.

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.10.025

Adleman has emphasized studying the number of steps it takes for an assembly to complete (assuming maximum parallelism) and the minimal number of tiles necessary to assemble a shape [4]. He answered these questions for n-long linear polymers. Here, I will extend these questions to apply to systems that compute functions, rather than assemble shapes.

Adleman proposed studying the complexity of tile systems that can uniquely produce $n \times n$ squares. A series of researchers proceeded to answer the questions: "What is a minimal tile set that can assemble such shapes?" and "What is the assembly time for these systems?" They showed that the minimal tile set is of size $\Theta(\frac{\log n}{\log \log n})$ and the optimal assembly time is $\Theta(n)$ [19,2,5]. A key issue related to assembling squares is the assembly of small binary counters, which theoretically can have as few as seven tile types [13].

While the constructions in this paper are in some ways analogous to traditional computer programs, and their running times are polynomially related to the running times of Turing machines, Baryshnikov et al. began the study of fundamental limits on the time required for a self-assembly system to compute functions [8]. They consider models of molecular self-assembly and apply Markov models to show lower limits on assembly times.

Researchers have also studied variations of the traditional tile assembly model. Aggarwal et al. and Kao et al. have shown that changing the temperature of assembly from a constant throughout the assembly process to a discrete function reduces the minimal tile set that can build an $n \times n$ square to a size $\Theta(1)$ tile set [1,15].

Barish et al. have demonstrated a DNA implementation of a tile system that copies an input and counts in binary [11]. Similarly, Rothemund et al. have demonstrated a DNA implementation of a tile system that computes the *xor* function, resulting in a Sierpinski triangle [17]. These systems grow crystals using double-crossover complexes [14] as tiles. The theoretical underpinnings of these systems are closely related to the work presented here because these systems compute functions, although in both cases these systems have a single input (*xor* is computed on consecutive bits of the input), while my systems have two inputs.

All systems presented here use a number of components polynomial in the input to solve computational problems. Some experimental work [3,7] has shown that it is possible to work with an exponential number of components. It is feasible to consider extending my results presented here to explore solving NP-complete problems or other problems for which we do not know of polynomial-time algorithms.

The rest of this paper is structured as follows: Section 1.1 will describe in detail the tile assembly model, Section 2 will describe and analyze systems that compute the sum and product of numbers, and Section 3 will summarize the contributions of this work.

1.1. Tile assembly model

The tile assembly model [23,19] is a formal model of crystal growth. It was designed to model self-assembly of molecules such as DNA. It is a partial extension of a model proposed by Wang [20]. The model was fully defined by Rothemund and Winfree [19], but I will restate some useful definitions here to assist the reader. Intuitively, the model has *tiles* or squares that stick or do not stick together based on various binding domains on their four sides. Each tile has a binding domain on its north, east, south, and west side, and may stick to another tile when the binding domains on the abutting sides of those tiles match and the total strength of all the binding domains on that tile exceeds the current temperature. The four binding domains define the type of the tile. While this definition does not allow tiles to rotate, it is essentially equivalent to a system with rotating tiles.

Formally, let Σ be a finite alphabet of binding domains such that $null \in \Sigma$. A **tile** over a set of binding domains Σ is a 4-tuple $\langle \sigma_N, \sigma_E, \sigma_S, \sigma_W \rangle \in \Sigma^4$. A **position** is an element of $\mathbb{Z}^2$. The set of directions $D = \{N, E, S, W\}$ is a set of four functions from positions to positions, i.e. $\mathbb{Z}^2$ to $\mathbb{Z}^2$ such that for all positions (x, y), $N(x, y) = (x, y + 1)$, $E(x, y) = (x + 1, y)$, $S(x, y) = (x, y - 1)$, $W(x, y) = (x - 1, y)$. The positions (x, y) and (x', y') are neighbors iff $\exists d \in D$ such that $d(x, y) = (x', y')$. For a tile t, for $d \in D$, I will refer to $bd_d(t)$ as the binding domain of tile t on d's side. A special tile $empty = \langle null, null, null, null \rangle$ represents the absence of all other tiles.

A **strength function** $g : \Sigma \times \Sigma \to \mathbb{R}$, where g is commutative and $\forall \sigma \in \Sigma$, $g(null, \sigma) = 0$, denotes the strength of the binding domains. It is common to assume that $g(\sigma, \sigma') = 0 \iff \sigma \neq \sigma'$. This simplification of the model implies that the abutting binding domains of two tiles have to match to bind. For the remainder of this paper, I will use $g = 1$ to mean $\forall \sigma \neq null\ g(\sigma, \sigma) = 1$ and $\forall \sigma' \neq \sigma\ g(\sigma, \sigma') = 0$.

Let T be a set of tiles containing the empty tile. A **configuration** of T is a function $A : \mathbb{Z} \times \mathbb{Z} \to T$. I write $(x, y) \in A$ iff $A(x, y) \neq empty$. A is finite iff there is only a finite number of distinct positions $(x, y) \in A$.

Finally, a **tile system** $\mathbb{S}$ is a triple $\langle T, g, \tau \rangle$, where T is a finite set of tiles containing *empty*, g is a strength function, and $\tau \in \mathbb{N}$ is the temperature.

If A is a configuration, then within system $\mathbb{S}$, a tile t can **attach** to A at position (x, y) and produce a new configuration A' iff:

- $(x, y) \notin A$, and
- $\sum_{d \in D} g(bd_d(t), bd_{d^{-1}}(A(d(x, y)))) \geq \tau$, and
- $\forall (u, v) \in \mathbb{Z}^2, (u, v) \neq (x, y) \Rightarrow A'(u, v) = A(u, v)$, and
- $A'(x, y) = t$.

That is, a tile can attach to a configuration only in empty positions and only if the total strength of the appropriate binding domains on the tiles in neighboring positions meets or exceeds the temperature τ. For example, if $g = 1$ and $\tau = 2$ then a tile t can attach only at positions with matching binding domains on the tiles in at least two adjacent positions.

Given a tile system $\mathbb{S} = \langle T, g, \tau \rangle$, a set of tiles Γ, and a **seed configuration** $S : \mathbb{Z}^2 \to \Gamma$, if the above conditions are satisfied, one may attach tiles of T to S. Configurations produced by repeated attachments of tiles from T are said to be **produced** by $\mathbb{S}$ on S. If this process terminates, then the configuration achieved when no more attachments are possible is called the **final configuration**. At some times, it may be possible for more than one tile to attach at a given position, or there may be more than one position where a tile can attach. If for all sequences of tile attachments, all possible final configurations are identical, then $\mathbb{S}$ is said to produce a **unique** final configuration on S.

Let $\mathbb{S} = \langle T, g, \tau \rangle$, and let S_0 be a seed configuration such that $\mathbb{S}$ produces a unique final configuration F on S_0. Let $W_0 \subseteq 2^{T \times \mathbb{Z}^2}$ be the set of all tile–position pairs $\langle t, (x, y) \rangle$ such that t can attach to S_0 at (x, y). Let S_1 be the configuration produced by adding all the elements of W_0 to S_0 in one time step. Define $W_1, W_2, \ldots$ and $S_2, S_3, \ldots$ similarly. Let n be the smallest natural number such that $S_n \equiv F$. Then n is the **assembly time** of $\mathbb{S}$ on S_0 to produce F.

I allow the codomain of S to be Γ, a set of tiles which may be different from T. The reason is that I will study systems that compute functions using minimal sets T; but the seed, which has to code for the input of the function, may contain more distinct tiles than there are in T. Therefore, I wish to keep the two sets separate. Note that, at any temperature τ, it takes $\Theta(n)$ distinct tiles to assemble an arbitrary n-bit input such that each tile codes for exactly one of the bits.

Winfree has shown that the tile assembly model with $\tau = 2$ is Turing-universal [23] by showing that a tile system can simulate Wang tiles [20], which Robinson has shown to be universal [16]. Adleman et al. have shown that the tile assembly model with $\tau = 1$ is Turing-universal [6].

2. Adder and multiplier

Intuitively, in order for a tile system to compute a function $f : \mathbb{N} \to \mathbb{N}$, if the seed encodes a number $a \in \mathbb{N}$ then the tile system should produce a unique final configuration which encodes $f(a)$. (Note that one may define a notion of non-deterministic computation that does not require a unique final configuration, instead taking advantage of non-determinism in the tile system.) To allow configurations to encode numbers, I will designate each tile as a 1 tile or a 0 tile and define an ordering on the positions of the tiles. The ith bit of a is 1 iff the tile in the ith position of the seed configuration is a 1 tile (note that *empty* will almost always be a 0 tile). I will also require that the seed contains tiles coding for bits of a and no more than a constant number of extraneous tiles. The final configuration may have an alternative ordering of the positions. Formally, let Γ and T be sets of tiles. Let Δ be the set of all possible seed configurations $S : \mathbb{Z}^2 \to \Gamma$, and let Ξ be the set of all possible finite configurations $C : \mathbb{Z}^2 \to \Gamma \cup T$. Let $v : \Gamma \cup T \to \{0, 1\}$ code each tile as a 1 tile or a 0 tile and let $o_s, o_f : \mathbb{N} \to \mathbb{Z}^2$ be two injections. Let the seed encoding function $e_s : \Delta \to \mathbb{N}$ map a seed S to a number such that $e_s(S) = \sum_{i=0}^{\infty} 2^i v(S(o_s(i)))$ if and only if for no more than a constant number of (x, y) not in the image of o_s, $(x, y) \in S$. Let the answer encoding function $e_f : \Xi \to \mathbb{N}$ map a configuration F to a number such that $e_f(F) = \sum_{i=0}^{\infty} 2^i v(F(o_f(i)))$.

Let $\mathbb{S} = \langle T, g, \tau \rangle$ be a tile system. I say that $\mathbb{S}$ computes a function $f : \mathbb{N} \to \mathbb{N}$ iff for all $a \in \mathbb{N}$ there exists a seed configuration S such that $\mathbb{S}$ produces a unique final configuration F on S and $e_s(S) = a$ and $e_f(F) = f(a)$.

By a similar approach, I define a tile system that can compute a function of two variables. Let $o_{s_1}, o_{s_2} : \mathbb{N} \to \mathbb{Z}^2$ be two injections. Let the seed encoding functions $e_{s_1}, e_{s_2} : \Delta \to \mathbb{N}$ map a seed S to a number such that

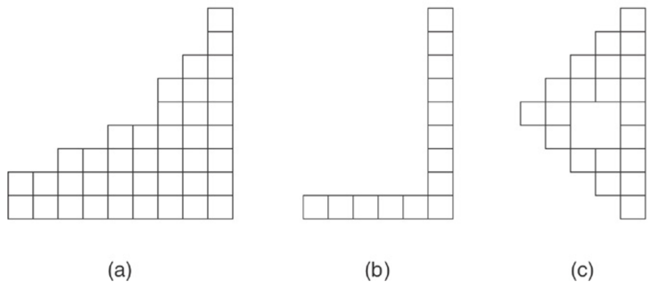


Fig. 1. A step-configuration (a), an L-configuration (b), and a configuration that is neither a step-configuration nor an L-configuration (c). A step-configuration has no holes and consists of a finite number of consecutive rows, such that each row has the same rightmost position and is shorter than the row below it. An L-configuration is a special case of a step-configuration and looks like the mirror image of the letter L.

$e_{s_1}(S) = \sum_{i=0}^{\infty} 2^i v(S(o_{s_1}(i)))$ and $e_{s_2}(S) = \sum_{i=0}^{\infty} 2^i v(S(o_{s_2}(i)))$ iff for no more than a constant number of (x, y) not in the union of the images of o_{s_1} and o_{s_2}, $(x, y) \in S$. Then I say that $\mathbb{S}$ computes a function $f : \mathbb{N}^2 \to \mathbb{N}$ iff for all $a_1, a_2 \in \mathbb{N}$ there exists a seed configuration S such that $\mathbb{S}$ produces a unique final configuration F on S and $e_{s_1}(S) = a_1$ and $e_{s_2}(S) = a_2$ and $e_f(F) = f(a_1, a_2)$.

In the remainder of this paper I will examine adders, systems that compute $f(a, b) = a + b$; and multipliers, systems that compute $f(a, b) = ab$. I require adder and multiplier systems to encode their inputs in binary. I could ask each input number to have a unique tile ($|\Gamma| = \Theta(2^n)$, for n-bit inputs) and a unique tile for every possible pair of inputs ($|T| = \Theta(2^n)$). Such a system would compute in $\Theta(1)$ time. However, in implementing such a system one would need exponentially many different types of tiles (e.g. molecules), which would make the problem intractable. I choose, as is common in computer science, to look at systems that limit the number of components ($|\Gamma| + |T|$) to be polynomial in the size of the input. In most of the constructions presented here (all but one), $|\Gamma| = \Theta(1)$. I will present systems with small constants and show their running time is linear in the input size.

2.1. Tile system configurations

In this section, I will prove three lemmas that will later be useful in studying tile systems.

Let A be a configuration. A is a *step-configuration* iff $\exists y_{\min}, y_{\max}, x_{\max} \in \mathbb{Z}$ such that all of the following are true:

- $\forall x, y \in \mathbb{Z}$ such that $y < y_{\min}$ or $y > y_{\max}$, $(x, y) \notin A$, and
- $\forall x, y \in \mathbb{Z}$ such that $x > x_{\max}$, $(x, y) \notin A$, and
- $\forall x, y \in \mathbb{Z}$, $\exists k_y \in \mathbb{N}$ such that:
  - . $k_y \le k_{y-1}$, and
  - . $x \le x_{\max} - k_y \Rightarrow (x, y) \notin A$, and
  - . $x_{\max} - k_y < x \le x_{\max} \Rightarrow (x, y) \in A$.

Intuitively, a step-configuration has no holes and consists of a finite number of consecutive rows such that each row has the same rightmost position and is no longer than the row below it. Fig. 1(a) and (b) show examples of a step-configurations and Fig. 1(c) shows an example of a configuration that is **not** a step-configuration.

Let A be a configuration. A is an *L-configuration* iff $\exists y_{\min}, y_{\max}, x_{\min}, x_{\max} \in \mathbb{Z}$ such that $y_{\min} \le y \le y_{\max}$ and $x_{\min} \le x \le x_{\max} \iff (x, y_{\min}) \in A$ and $(x_{\max}, y) \in A$. Intuitively, an L-configuration looks like a mirror image of the letter L. Clearly, an L-configuration is a step-configuration. Fig. 1(b) shows an example of an L-configuration.

Lemma 2.1 (*Step Configuration Lemma*). *Let $\mathbb{S} = \langle T, g, \tau \rangle$ be a tile system such that $g = 1$ and $\tau = 2$. Let Γ be a set of tiles and let $S : \mathbb{Z}^2 \to \Gamma$ be a seed step-configuration. All configurations produced by $\mathbb{S}$ on S are step-configurations.*

Proof (*By Induction over Tile Addition*). The base case is trivial because the seed configuration is assumed to be a step-configuration. Now, assume that you have a step-configuration A and add the next tile to produce A'. Since all the binding domains are of strength 1, and $\tau = 2$ a tile may only attach if it is adjacent to τ or more tiles in A. Thus possible places to attach are "corners" of A, or places at the end of a non-empty row that have a longer row below it. That is, a tile may attach at (x, y) iff $(x, y) \notin A$ and $(x + 1, y) \in A$ and $(x, y - 1) \in A$. Therefore, $k_{y-1} > k_y$ and since only one tile is added, after the addition $k_{y-1} \ge k_y$. Therefore A' is a step-configuration. $\square$

Lemma 2.2 (*Unique Final Configuration Lemma*). *Let $\mathbb{S} = \langle T, g, \tau \rangle$ be a tile system such that $g = 1$ and $\tau = 2$. Let Γ be a set of tiles and let $S : \mathbb{Z}^2 \to \Gamma$ be a seed step-configuration. If $\forall t \in T$ the pair $(bd_s(t), bd_e(t))$ is unique then $\mathbb{S}$ produces a unique final configuration on S.*

Proof. First of all, because $\tau = 2$ and $g = 1$, a tile may only attach if it has two neighbors, and thus the configuration will never grow larger than the bounding box of S, so there will be a final configuration. Second, by Lemma 2.1, whenever a tile attaches, it will attach to a step-configuration, and thus the only positions at which it can attach will have the south and east neighbors available for binding. Since $\tau = 2$ both of them have to be available, and since the pair $(bd_S(t), bd_E(t))$ is unique for each tile t, only one type of tile may attach at every given (x, y) position. Therefore, the final configuration is unique. □

Corollary 2.1 (*Unique Final Configuration Corollary*). *Let $\mathbb{S} = \langle T, g, \tau \rangle$ be a tile system such that $g = 1$ and $\tau = 2$. Let Γ be a set of tiles and let $S : \mathbb{Z}^2 \to \Gamma$ be a seed L-configuration but with the corner south east tile missing and let the binding domains neighboring that corner be null. Then if $\forall t \in T$, the pair $(bd_s(t), bd_e(t))$ is unique then $\mathbb{S}$ produces a unique final configuration on S.*

Proof. Since the binding domains neighboring the south east corner are *null*, no tile can attach there. By Lemma 2.2, the rest of the seed configuration produces a unique final configuration. Thus $\mathbb{S}$ produces a unique final configuration on S. □

Lemma 2.3 (*Assembly Time Lemma*). *Let $\mathbb{S} = \langle T, g, \tau \rangle$ be a tile system such that $g = 1$ and $\tau = 2$. Let Γ be a set of tiles, let $S : \mathbb{Z}^2 \to \Gamma$ be a seed L-configuration, and let $x_{\max}$ and $y_{\max}$ be the lengths of the bottom row and rightmost column of S, respectively. If $\forall t \in T$ the pair $(bd_s(t), bd_e(t))$ is unique then $\mathbb{S}$ produces a unique final configuration on S. If that final configuration is the complete $x_{\max} \times y_{\max}$ rectangle, then the assembly time is $\Theta(x_{\max} + y_{\max})$.*

Proof. $\mathbb{S}$ produces a unique final configuration on S by Lemma 2.2. Let that configuration be F. Assume F is the complete $x_{\max}$ by $y_{\max}$ rectangle. Without loss of generality, assume $x_{\max} \geq y_{\max}$. It is clear that, at first, only a single tile may attach to S, in the corner position. After that tile attaches, there are two new corner positions that may have a tile attach. And so on, for the first $y_{\max}$ time steps, during the ith time step exactly i tiles may attach. After that, for the next $x_{\max} - y_{\max}$ time steps, exactly $y_{\max}$ tiles may attach per step. Finally, for the last $y_{\max}$ steps, on the jth time step, exactly $y_{\max} - j$ tiles may attach. The resulting configuration is the $x_{\max} \times y_{\max}$ rectangle, and the assembly time is $y_{\max} + x_{\max}$ steps. □

Corollary 2.2 (*Assembly Time Corollary*). *Let $\mathbb{S} = \langle T, g, \tau \rangle$ be a tile system such that $g = 1$ and $\tau = 2$. Let Γ be a set of tiles, let $S : \mathbb{Z}^2 \to \Gamma$ be a seed L-configuration but with a single tile missing: the corner tile and let the binding domains neighboring that corner be null, and let $x_{\max}$ and $y_{\max}$ be the lengths of the bottom row and rightmost column of S. If $\forall t \in T$ the pair $(bd_s(t), bd_e(t))$ is unique then, $\mathbb{S}$ produces a unique final configuration on S. If that final configuration is the (almost) complete $x_{\max} \times y_{\max}$ rectangle (missing only the same single corner as S), then the assembly time is $\Theta(x_{\max} + y_{\max})$.*

Proof. By Corollary 2.1, $\mathbb{S}$ produces a unique final configuration on S. By Lemma 2.3, the assembly time is $\Theta(x_{\max} + y_{\max})$. □

Let $a = \sum_i 2^i a_i$ where $a_i \in \{0, 1\}$. I will often say $bit_i(a)$ or the ith bit of a to mean a_i, so the indexing of the bits starts at 0. I will also say $cbit_i(a, b)$ or the ith carry bit of $a + b$ to mean 1 iff the sum of a_{i-1}, b_{i-1}, and $cbit_{i-1}(a, b)$ is greater than or equal to 2. Formally, $cbit_0(a, b) = 0$ and $cbit_i(a, b) = bit_1(a_{i-1} + b_{i-1} + cbit_{i-1}(a, b))$.

2.2. Adder tile systems

In this section, I will first discuss a small tile system that computes $f(x, y) = x + y$ using $\Theta(1)$ distinct tiles, I will second show a system that uses $\Theta(n)$ tiles but presents important ideas, and finally, I will introduce a new concept to simplify that system to use only $\Theta(1)$ tiles.

For some tile systems and some seeds, tiles will attach only when certain neighbors are present. I call the sides of the tile that attach to these neighbors the "input" sides. I call the other sides the "output" sides. For example, given a tile system $\langle T, 1, 2 \rangle$ and a step seed configuration, tiles will only attach to a configuration at a position if the east and

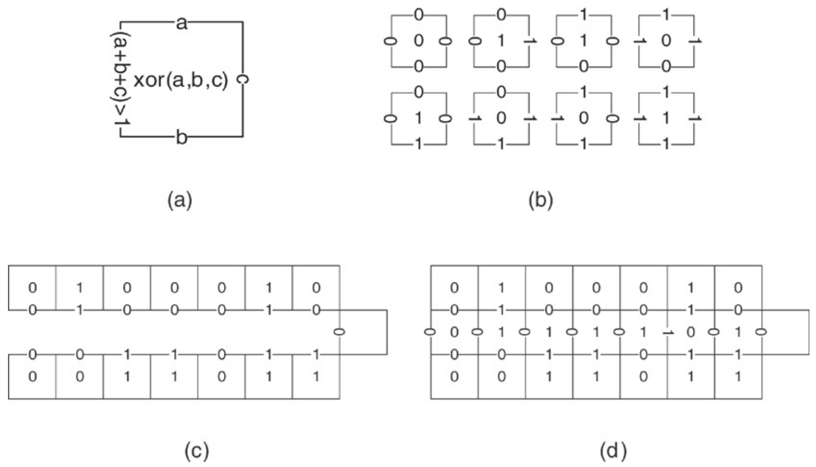


Fig. 2. Tile system $\mathbb{S}_{+8}$ computes the function $f(a, b) = a + b$. The tiles have three input sides (east, north, and south) and one output side (west) (a). Each tile is labeled with a 1 or a 0 to assist the reader with reading the encoding v_8. There are eight tiles in the system (b). Given a sample input of $b = 100010_2 = 34$ and $a = 11011_2 = 27$ (c), with 34 on top row and 27 on the bottom row, the system fills the row in the middle with $a + b = 111101_2 = 61$ to produce the unique final configuration (d). Note that the least significant digit is on the right. Also note that the inputs are padded with extra 0 tiles in the most significant bit places because the sum of two n-bit numbers may be as large as $n + 1$ bits.

south neighbors of that position are in the configuration. The east and south sides of these tiles are the input sides and the west and north sides are the output sides. The input sides determine which tile attaches to a configuration now, and the output sides determine which tiles may attach later.

2.2.1. Eight-tile adder

Theorem 2.1. *Let $\Sigma_8 = \{0, 1\}$, $g_8 = 1$, $\tau_8 = 3$, and T_8 be a set of tiles over Σ_8 as described in Fig. 2(b). Then $\mathbb{S}_{+8} = \langle T_8, g_8, \tau_8 \rangle$ computes the function $f(a, b) = a + b$.*

Intuitively, $\mathbb{S}_{+8}$ has eight tiles with the east, north, and south sides as the input sides and the west side as the output side, outputting 1 iff the sum of the inputs is at least 2. Note that I do not formally define the tiles of T because it is easiest to read their descriptions from Fig. 2(b).

To encode the answer, the type of the tile is determined by the sum of the inputs, mod 2. Fig. 2(a) shows the concept behind the tiles, and Fig. 2(b) shows the eight tiles for all possible 0 and 1 binding domains for the three input sides. The 1 or 0 in the middle of the tile t is that tile's $v_8(t)$ value.

Fig. 2(c) shows a sample seed configuration which encodes two numbers in binary: $100010_2 = 34$, $11011_2 = 27$. Number 34 is encoded on the top row and number 27 is encoded on the bottom row. There are five tiles in Γ_8, the 1 and 0 tiles for each of the two inputs, and the single starter tile on the right side. Note that at the start, only one tile may attach to this configuration because $\tau_8 = 3$. Fig. 2(d) shows the final configuration for the example of $34 + 27$, with the solution encoded on the center row. The row reads 111101_2 which is $61 = 34 + 27$.

Because the sum of two n-bit numbers may be as large as $n + 1$ bits, each of the two inputs needs to be padded to be $n + 1$ bits long with extra 0 tiles.

Proof of Theorem 2.1. Consider the tile system $\mathbb{S}_{+8}$. Let a and b be the numbers to add and let n_a and n_b be the sizes, in bits, of a and b, respectively. Let $n = \max(n_a, n_b)$. For all $i \in \mathbb{N}$ and let $a_i, b_i \in \{0, 1\}$ such that $a = \sum_i 2^i a_i$ and $b = \sum_i 2^i b_i$.

Let $\Gamma_8 = \{\alpha_0 = \langle 0, \mathit{null}, \mathit{null}, \mathit{null} \rangle, \alpha_1 = \langle 1, \mathit{null}, \mathit{null}, \mathit{null} \rangle, \beta_0 = \langle \mathit{null}, \mathit{null}, 0, \mathit{null} \rangle, \beta_1 = \langle \mathit{null}, \mathit{null}, 1, \mathit{null} \rangle, \gamma = \langle \mathit{null}, \mathit{null}, \mathit{null}, 0 \rangle\}$. Let the seed configuration $S : \mathbb{Z}^2 \to \Gamma_8$ be such that

- $S(1, 0) = \gamma$,
- $\forall i = 0, \ldots, n$, $S(-i, -1) = \alpha_{a_i}$,
- $\forall i = 0, \ldots, n$, $S(-i, 1) = \beta_{b_i}$, and
- for all other $(x, y) \in \mathbb{Z}^2$, $S(x, y) = \mathit{empty}$.

It is clear that there is only a single position where a tile may attach to S. It is also clear that after that tile attaches there will only be a single position where a tile may attach. By induction, because $\forall t \in T_8$ the triplet $\langle bd_S(t), bd_E(t), bd_N(t)\rangle$ is unique, and because $\tau_8 = 3$, it follows that $\mathbb{S}$ produces a unique final configuration on S. Let that configuration be F.

For all $0 \leq i \leq n$, S and F agree on $(-i, -1)$ and $(-i, 1)$. Therefore, $bd_N(F(-i, -1)) = a_i$ and $bd_S(F(-i, 1)) = b_i$. I will now show, by induction, that $v(F(-i, 0)) = bit_i(a + b)$ and $bd_W(F(-i, 0)) = cbit_{i+1}(a, b)$.

From the definition of S, $bd_W(F(1, 0))$ is $0 = cbit_0(a, b)$. Now I assume that $bd_W(F(-(i - 1), 0)) = cbit_i(a, b)$ and show that $bd_W(F(-i, 0)) = cbit_{i+1}(a, b)$ and $v(F(-i, 0)) = bit_i(a + b)$. Let $t = F(-i, 0)$. The value $v(t)$ is $xor(bd_N(t), bd_E(t), bd_S(t))$. Since t binds with strength 3, $bd_N(t) = bd_S(F(-i, 1))$, $bd_E(t) = bd_W(F(-(i-1), 0))$, $bd_S(t) = bd_N(F(-i, -1))$, so $v(t)$ is the xor of a_i, b_i, and $cbit_i(a, b)$. The ith bit of $a + b$ is exactly that xor. The binding domain $bd_W(t)$ is defined as 1 if at least two of a_i, b_i, and $cbit_i(a, b)$ are 1 and 0 otherwise. That is exactly the definition of $cbit_{i+1}(a, b)$.

Thus, for all $1 \leq i \leq n$, $v(F(-i, 0)) = bit_i(a + b)$. It is clear that those are the only tiles of T in F. For all $j \in \mathbb{N}$, let $o_{s_1}(j) = (-j, -1)$, $o_{s_2}(j) = (-j, 1)$, and $o_f(j) = (-j, 0)$. Because $a + b$ is no longer then $n + 1$ bits, it follows that $\mathbb{S}_{+8}$ computes $f(a, b) = a + b$. □

The logic of the system is identical to a series of one-bit full adders. Each solution tile takes in a bit from each of the inputs on the north and south sides and a carry bit from the previous solution tile on the east side, and outputs the next carry bit on the west side. Because $\tau_8 = 3$, only a tile with three neighbors may attach at any time, and therefore, no tile may attach until its right neighbor has. Thus the assembly time for this system is n steps to add two n-bit numbers. Note that $|\Gamma_8| = 5$ and $|T_8| = 8$.

2.2.2. L-configuration adder

I now present an adder tile system that uses $\Theta(n)$ tile types to compute, but builds on L-shape seed configurations. This adder will use the two sides of the L-configuration to encode inputs, and produce its output on the top row of an almost complete rectangle. Therefore, systems could chain computations together, using the output of this computation as an input to another computation.

Theorem 2.2. *For all $n \in \mathbb{N}$, let $\Sigma_n = \{0, 1\} \cup \{\#i | i = 0, 1, \ldots, n\}$, $g_n = 1$, $\tau_n = 2$, and T_n be a set of tiles over Σ_n as described in Fig. 3*(b). *Then $\mathbb{S}_{+n} = \langle T_n, g_n, \tau_n\rangle$ computes the function $f(a, b) = a + b$ for all $a, b < 2^n$.*

Let n_a and n_b be the sizes, in bits, of a and b, respectively, and let $n = \max(n_a, n_b)$. Intuitively, $\mathbb{S}_{+n}$ adds a and b using $\Theta(n)$ distinct tile types. One of the input numbers, a, is coded on the bottom row and the second input number, b, on the rightmost column of an L-configuration. The adder adds one bit of b to a in each row. The ith bit has to be added in the ith column, and the system uses the $\Theta(n)$ tiles to count down to the correct position. Each tile has two input sides (south and east) and two output sides (north and west). The north side is the value of the tile and the west side is the carry bit. The tile descriptions are easiest to read from Fig. 3(b).

To encode the answer, the type of the tile is determined by the sum of the inputs, mod 2. Fig. 3(a) shows the concepts behind the tiles, and Fig. 3(b) shows the actual tiles (but only some of the counting tiles) for $n \geq 99$. The 1 or 0 in the middle of the tile t is that tile's $v(t)$ value.

Fig. 3(c) shows a sample seed configuration which encodes two numbers in binary: $1010111_2 = 87$, $101101_2 = 45$. Number 87 is encoded on the bottom row and number 45 is encoded on the column. There are up to $n + 3$ tiles in Γ_n, the 1 and 0 tiles for the row input, the 0 tile for the column input, and up to n distinct tiles for the 1 bits of the column input. Note that at the start, only one tile may attach to this configuration because $\tau_n = 2$ and there is only a single corner. By Corollary 2.1, $\mathbb{S}_{+n}$ produces a unique final configuration. Fig. 3(d) shows the final configuration for the example of $87 + 45$, with the solution encoded on the top row. The row reads 10000100_2 which is $132 = 87 + 35$.

Again, because the sum of two n-bit numbers may be as large as $n + 1$ bits, the row input needs to be padded with a single extra 0 tile as its most significant bit. The column input does not need this padding.

Each row adds a single bit of the column input at the correct location. Theorem 2.2 follows from the logic of the tiles (I do not bother to formally prove it here because the purpose of this system is to display ideas useful in designing later systems). By Corollary 2.2, the assembly time for adding two n-bit numbers is $\Theta(n)$ steps. Note that $|\Gamma_n| = n+3$ and $|T_n| = 2n + 6$.

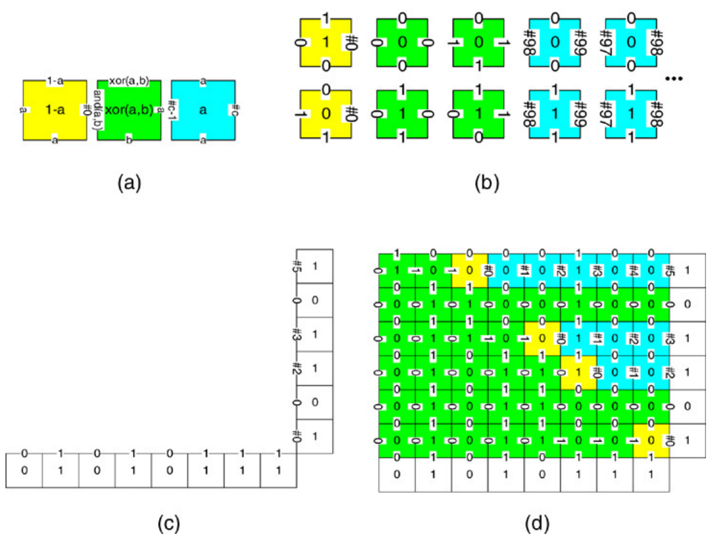


Fig. 3. Tile system $\mathbb{S}_{+n}$ computes the function $f(a, b) = a + b$ and uses $\Theta(n)$ distinct tile types. The tiles have two input sides (east and south) and two output sides (west and north). The north side is the value of the bit, and the east side is the carry bit (a). Each tile is labeled with a 1 or a 0 to assist the reader with reading the encoding v. There are $2n + 6$ tiles in the system (b). Given a sample input of $a = 1010111_2 = 87$, and $b = 101101_2 = 45$ (c), with 87 on the bottom row and 45 on the rightmost column, the system fills the rectangle in the middle by adding a single bit of b in each row. The top row reads the solution: $a + b = 10000100_2 = 132$. Note that the least significant digit is on the right and that the color is purely to help the reader track tiles; the tiles themselves have no sense of color. Also note that the row input a has an extra 0 tile in the most significant bit place because the sum of two n-bit numbers may be as large as $n + 1$ bits. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

2.2.3. L-configuration constant-tile adder

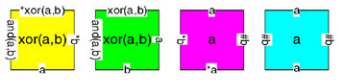
I now modify the adder that acts on L-configurations to only use $\Theta(1)$ tiles to compute by invoking an idea of sandwiching tiles, which will be important in constructing a multiplier system. As before, this adder will produce its output on the top row of a rectangle and allow the chaining of computations.

Theorem 2.3. *Let $\Sigma_{16} = \{0, 1, \#0, \#1, *0, *1\}$, $g_{16} = 1$, $\tau_{16} = 2$, and T_{16} be a set of tiles over Σ_{16} as described in Fig. 4*(b). *Then $\mathbb{S}_{+16} = \langle T_{16}, g_{16}, \tau_{16}\rangle$ computes the function $f(a, b) = a + b$.*

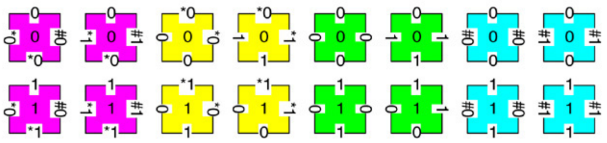
Intuitively, $\mathbb{S}_{+16}$ adds two n-bit numbers using 16 tiles. While the previous adder used i tiles to count down to the ith position, this tile system will use a constant number of tiles to find that position. It is easy to imagine how to create a tile system that finds the ith position in the ith row (or just builds a diagonal line). The key to $\mathbb{S}_{+16}$ is to make one system compute both, the diagonal line and the sum, by using a technique related to tile-sandwiching. A sandwich tile [6] is a tile which is essentially made up of two tiles. Suppose a set of tiles A over Σ_A constructs a diagonal line and a set of tiles B over Σ_B ensures that red tiles only attach to blue tiles on the east–west sides. Then one can create a set of sandwich tiles C over $\Sigma_A \times \Sigma_B$ which will make striped diagonal lines. C is defined such that if $a = \langle\sigma_{a,N}, \sigma_{a,E}, \sigma_{a,S}, \sigma_{a,W}\rangle \in A$ and $b = \langle\sigma_{b,N}, \sigma_{b,E}, \sigma_{b,S}, \sigma_{b,W}\rangle \in B$, then the sandwich tile of a and b is $c = \langle(\sigma_{a,N}, \sigma_{b,N}), (\sigma_{a,E}, \sigma_{b,E}), (\sigma_{a,S}, \sigma_{b,S}), (\sigma_{a,W}, \sigma_{b,W})\rangle \in C \subseteq (\Sigma_A \times \Sigma_B)^4$. In other words, if c is the sandwich of two tiles a and b and c' is the sandwich tile of a' and b', then c binds to c' iff a binds to a' and b binds to b'. Note that $|C| = |A||B|$.

$\mathbb{S}_{+16}$ uses the idea of sandwich tiles by sandwiching tiles that build a diagonal line with the tiles from $\mathbb{S}_{+n}$. First of all, note that a system with two tiles can form an infinite diagonal line. Though I do not formally describe such a system, I will call the set of two tiles that build a diagonal line T_{diagonal}. (The reader may examine the yellow and magenta tiles in Fig. 4, which under the right conditions would build a diagonal line, although they do more than just that.)

Remember that $\mathbb{S}_{+n}$ had three different kinds of tiles: yellow tiles that added the ith bit to the running sum, green tiles that added the number below to a possible carry bit, and blue tiles that counted down to the ith position and also propagated information up. Suppose I change this set of tiles so that the blue tiles no longer count down to the ith position, but the tile in that position is always yellow. Then I would need four blue tiles and four yellow tiles (and the same four green tiles as before) for a total of twelve tiles. I have now almost completely designed T_{16}; however, I have not explained how to get the yellow tiles to form the diagonal line. To that end, I sandwich the four yellow tiles



(a)


(b)

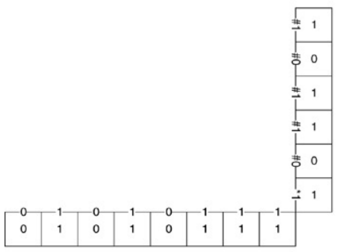
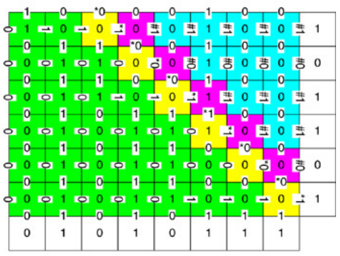


(c) (d)

Fig. 4. Tile system $\mathbb{S}_{+16}$ computes the function $f(a, b) = a + b$ and uses 16 distinct tile types. The tiles have two input sides (east and south) and two output sides (west and north). The north side is the value of the bit, and the east side is the carry bit (a). Each tile is labeled with a 1 or a 0 to assist the reader with reading the encoding v_{16}. There are 16 tiles in the system (b). Given a sample input of $a = 1010111_2 = 87$, and $b = 101101_2 = 45$ (c), with 87 on bottom row and 45 on the rightmost column, the system fills the rectangle in the middle by adding a single bit of y in each row. The top row reads the solution: $a + b = 10000100_2 = 132$. Note that the least significant digit is on the right and that the color is purely to help the reader track tiles; the tiles themselves have no sense of color. Also note that the row input a has an extra 0 tile in the most significant bit place because the sum of two n-bit numbers may be as large as $n + 1$ bits. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

with the two tiles in T_{diagonal} to create eight tiles, represented by yellow and magenta in Fig. 4(b). The union of green, blue, yellow, and magenta tiles makes up T_{16}.

In other words, in addition to the green, blue, and yellow tiles, $\mathbb{S}_{+16}$ also has magenta tiles that perform two jobs: propagating the information up just as the blue tiles and guiding the yellow diagonal line. Just as before, one of the input numbers is coded on the bottom row and the second input number on the rightmost column of an L-configuration. The adder adds one bit of the column number to the row number, per row. The ith bit has to be added at the ith position, and the system uses the yellow diagonal line to compute that position. Each tile has two input sides (south and east) and two output sides (north and west). The north side is the value of the tile and the west side is the carry bit. The tile descriptions are easiest to read from Fig. 4(b).

For answer encoding purposes, the type of the tile is determined by the sum of the inputs, mod 2. Fig. 4(a) shows the concepts behind the tiles, and Fig. 4(b) shows the actual 16 tiles. The 1 or 0 in the middle of the tile t is that tile's $v(t)$ value.

Fig. 4(c) shows a sample seed configuration which encodes two numbers in binary: $1010111_2 = 87$, $101101_2 = 45$, just as before. Number 87 is encoded on the bottom row and number 45 is encoded on the rightmost column. There are four tiles in Γ_{16}, the 1 and 0 tiles for each of the inputs. Note that at the start, only one tile may attach to this configuration because $\tau_{16} = 2$ and there is only a single corner. By Corollary 2.1, $\mathbb{S}_{+16}$ produces a unique final configuration. Fig. 4(d) shows the final configuration for the example of 87 + 45, with the solution encoded on the top row. The row reads 10000100_2 which is $132 = 87 + 35$. Note that in addition to the sum, the final construction "computes" a diagonal yellow (and also a magenta) line. Just as before, because the sum of two n-bit numbers may be as large as $n + 1$ bits, the row input needs to be padded with a single extra 0 tile as its most significant bit. The column input does not need this padding.

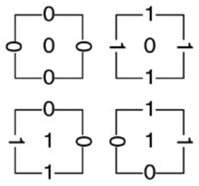


Fig. 5. A set of tiles that performs a left-shift on a row of 0s and 1s. In binary, a left-shift is a multiplication by 2.

Each row adds a single bit of the column input at the correct location. Theorem 2.3 follows from the logic of the tiles (I do not bother to formally prove it here because the purpose of this system is to display ideas useful in designing the next system, and I will prove that system's correctness formally). By Corollary 2.2, the assembly time for adding two n-bit numbers is $\Theta(n)$ steps. Note that $|\Gamma_{16}| = 4$ and $|T_{16}| = 16$.

2.3. *Multiplier tile system*

I have described two adder systems that start with an L-configuration and add a single bit per row to arrive at a row representing the sum of two numbers. I have also described briefly how to combine the functionality of two tile systems to produce a single tile system with properties of each of them. I will now illustrate how to combine these ideas to create a multiplier tile system. As with the last two adders, the multiplier will produce its output on the top row of a rectangle and allow the chaining of computations.

Observe that if $b = \sum_i b_i 2^i$, for $b_i \in \{0, 1\}$, that is, the ith bit of b is b_i, then the product ab can be written as $ab = \sum_i b_i a 2^i$. That is, one can multiply a by each of the powers of 2 in b, and then add up the products. Multiplying by two in binary is simple — it is a left-shift. Fig. 5 shows four tiles that would shift a row of 1 and 0 tiles to the left (I do not bother to define a formal system to do this, but just show the tiles). The tiles simply "flow" the information to the left and display the information received from the right. What is left is to add the rows representing the appropriate powers of 2 to construct the product of two numbers. The system should, therefore, add up to n numbers, as opposed to just 2. It is feasible to imagine such a system that on each row adds a new number to a running total, and arrives at the sum of n numbers on the nth row. Thus I have informally described two sets of tiles: one that performs a left-shift and one that adds a number to the running total on each row. Sandwiching these two sets of tiles produces a set of tiles that sums up products of some input number a and powers of 2, i.e. computes $\sum_i a2^i$. What is left is to add minor control to only sum up the appropriate powers of 2.

Theorem 2.4. *Let $\Sigma_\times = \{0, 1, 00, 01, 10, 11, 20, 21\}$, $g_\times = 1$, $\tau_\times = 2$, and $T_\times$ be a set of tiles over $\Sigma_\times$ as described in Fig. 6*(b). *Then $\mathbb{S}_\times = \langle T_\times, g_\times, \tau_\times \rangle$ computes the function $f(a, b) = ab$.*

Fig. 6 shows the tiles of $\mathbb{S}_\times$. Fig. 7 shows an example execution of $\mathbb{S}_\times$ on $a = 87$ and $b = 45$. The input a, as before, is encoded on the bottom row and the input b is encoded on the rightmost column. There are four special magenta tiles which deal with the input and fill in the bottom row of the computation. These tiles are necessary because the least significant bit of the column input is 2^0 and thus requires no left-shift. The blue tiles code rows that have a 0 in the input b. These tiles perform a left-shift, but do not add the current power of 2 to the running sum. The green and yellow tiles fill in the rows that have a 1 in the input b. Yellow tiles indicate the incoming carry bit is 0, and green tiles indicate that bit is 1. Each tile, in addition to its binding domains, is labeled with two pieces of information: the lower half of the tile on the ith row displays the appropriate bit of $2^i a$ and the upper half of the tile displays the appropriate bit of the running sum so far. The top half of the top row displays the solution. In the example, the solution is $0111101001011_2 = 3915 = ab$.

There are six tiles in $\Gamma_\times$, the 1 and 0 tiles for each of the inputs and special 0 and 1 tiles for the least significant bit of b. Note that at the start, only one tile may attach to this configuration because $\tau_\times = 2$ and there is only a single corner.

Before, the inputs to the adder systems had to be padded with a single 0 bit. Because the product of two n-bit numbers may be as large as $2n$ bits, the row input to the multiplier system needs to be padded with n extra 0 tiles. The column input does not need this padding.

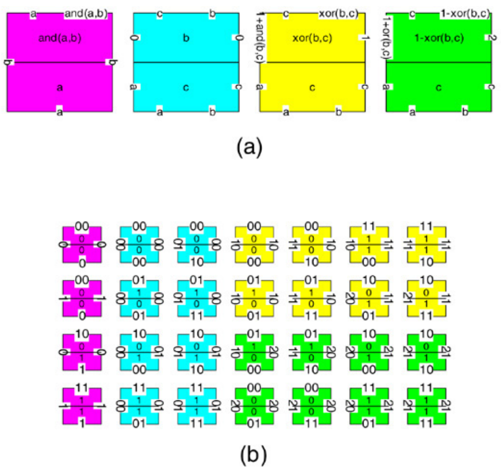


Fig. 6. Tile system $\mathbb{S}_\times$ computes the function $f(a,b) = ab$ and uses 28 distinct tile types. The tiles have two input sides (east and south) and two output sides (west and north) (a). There are 24 computation tiles and four special magenta tiles for starting the computation (b). Note that the colors are solely for the reader; the tiles themselves have no sense of color. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

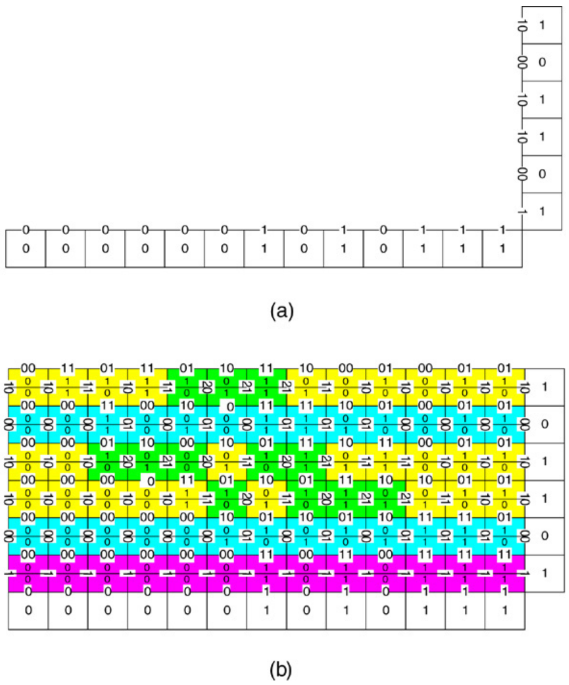


Fig. 7. An example of $\mathbb{S}_\times$. Given a sample input of $a = 1010111_2 = 87$, and $b = 101101_2 = 45$ (a), with 87 on bottom row and 45 on the rightmost column, the system fills the rectangle (b). The ith row has two pieces of information: on the lower half of each tile is the value of the appropriate bit of $2^i a$ and on the upper half is the running sum of the products of a and powers of 2 in b smaller than or equal to i. The upper portion of the top row reads the solution: $ab = 0111101001011_2 = 3915 = 87 \cdot 45$. Note that the least significant digit is on the right and that the color is purely to help the reader track tiles; the tiles themselves have no sense of color. Also note that the row input a has n_b extra 0 tiles in the most significant bit places because the product of two n-bit numbers may be as large as $2n$ bits. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

Proof of Theorem 2.4. Consider the tile system $\mathbb{S}_\times$. Let a and b be the numbers to multiply and let n_a and n_b be the sizes, in bits, of a and b, respectively. For all $i \in \mathbb{N}$ let $a_i, b_i \in \{0,1\}$ be such that $a = \sum_i 2^i a_i$ and $b = \sum_i 2^i b_i$.

Let $\Gamma_\times = \{\alpha_0 = \langle 0, \mathit{null}, \mathit{null}, \mathit{null}\rangle, \alpha_1 = \langle 1, \mathit{null}, \mathit{null}, \mathit{null}\rangle, \beta_0 = \langle \mathit{null}, \mathit{null}, \mathit{null}, 00\rangle, \beta_1 = \langle \mathit{null}, \mathit{null}, \mathit{null}, 10\rangle, *\beta_0 = \langle \mathit{null}, \mathit{null}, \mathit{null}, 0\rangle, *\beta_1 = \langle \mathit{null}, \mathit{null}, \mathit{null}, 1\rangle\}$. Then the seed configuration $S : \mathbb{Z}^2 \to \Gamma_\times$ is such that

- $\forall i = 0, \ldots, n_a + n_b - 1$, $S(-i, -1) = \alpha_{a_i}$,
- $S(1, 0) = *\beta_{b_0}$,
- $\forall i = 1, \ldots, n_b - 1$, $S(1, i) = \beta_{b_i}$, and
- for all other $(x, y) \in \mathbb{Z}^2$, $S(x, y) = \textit{empty}$.

By the unique final configuration corollary (Corollary 2.1), $\mathbb{S}_\times$ produces a unique final configuration on S. Let that configuration be F.

For those tiles with two-bit binding domains bd, let $l(bd)$ be the first bit and $r(bd)$ be second bit of the binding domain. For all $t \in T$, let $v(t) = r(bd_N(t))$. I will show that:

(1) $\forall 0 \le i < n_a + n_b$, $l(bd_N(F(-i, 0))) = a_i$ and $r(bd_N(F(-i, 0))) = a_i$ if $b_0 = 1$ and 0 otherwise.
(2) $\forall 0 \le i < n_a + n_b$, $\forall 1 \le j \le n_b - 1$, all the following are true:
- $l(bd_N(F(-i, j)))$ is the ith bit of $a2^j$,
- $r(bd_N(F(-i, j))) = bit_i(a(b \bmod 2^{j+1}))$,
- $l(bd_W(F(-i, j))) = \begin{cases} 0 & \text{if } b_j = 0 \\ 1 + cbit_{i+1}(a(b \bmod 2^j), a2^j) & \text{otherwise} \end{cases}$,
- $r(bd_W(F(-i, j))) = bit_i(a2^{j-1})$

Proof of (1) (by induction): For all $0 \le i < n_a + n_b$, S and F agree on $(-i, -1)$ and $(1, 0)$. Therefore, $bd_W(F(1, 0)) = b_0$. I assume that $bd_W(F(-(i-1), 0)) = b_{i-1}$ and I will show that $bd_W(F(-i, 0)) = b_i$, $l(bd_N(F(-i, 0))) = a_i$, and $r(bd_N(F(-i, 0))) = a_i$ if $b_0 = 1$ and 0 otherwise. Note that only magenta tiles may attach in this row because they are the only ones with a single-bit east binding domain. For all such tiles t', $bd_W(t') = bd_E(t')$, so $bd_W(t') = b_0$. Also, $l(bd_N(t')) = bd_S(t')$, and since $F(-i, -1) = S(-i, -1)$ and $bd_N(S(-i, -1)) = a_i$, it follows that $l(bd_N(t')) = a_i$. Finally, $r(bd_N(F(-i, 0))) = 1$ iff $bd_S(t') = 1$ and $bd_E(t') = 1$, so $r(bd_N(F(-i, 0))) = a_i$ if $b_0 = 1$ and 0 otherwise.

Proof of (2) (by induction): Base cases: $\forall 0 \le i < n_a + n_b$, $l(bd_N(F(-i, 0))) = a_i$ and $r(bd_N(F(-i, 0))) = a_i$ if $b_0 = 1$ and 0 otherwise (by (1)); $\forall 0 < j < n_b$, $l(bd_W(F(0, j))) = b_j$ and $r(bd_W(F(0, j))) = 0$. Inductive hypothesis: I assume that

- $l(bd_N(F(-i, j-1))) = bit_i(a2^{j-1})$,
- $r(bd_N(F(-i, j-1))) = bit_i(a(b \bmod 2^j))$,
- $l(bd_W(F(-(i-1), j))) = \begin{cases} 0 & \text{if } b_j = 0 \\ 1 + cbit_i(a(b \bmod 2^j), a2^j) & \text{otherwise} \end{cases}$,
- $r(bd_W(F(-(i-1), j))) = bit_{i-1}(a2^{j-1})$.

and show that

(i) $l(bd_N(F(-i, j))) = bit_i(a2^j)$,
(ii) $r(bd_N(F(-i, j))) = bit_i(a(b \bmod 2^{j+1}))$,
(iii) $l(bd_W(F(-i, j))) = \begin{cases} 0 & \text{if } b_j = 0 \\ 1 + cbit_{i+1}(a(b \bmod 2^j), a2^j) & \text{otherwise} \end{cases}$
(iv) $r(bd_W(F(-i, j))) = bit_i(a2^{j-1})$.

Let $t = F(-i, j)$. Note that t cannot be magenta because for all $j \ge 1$ only non-magenta tiles may attach.

(i): For all non-magenta tiles in position $(-i, j)$:

$$\begin{aligned} l(bd_N(F(-i, j))) &= r(bd_E(F(-i, j))) \\ &= r(bd_W(F(-(i-1), j))) \\ &= bit_{i-1}(a2^{j-1}) \\ &= bit_i(a2^j). \end{aligned}$$

(ii) and (iii): There are three possible cases:

(1) ($b_j = 0$): only a blue tile may attach in position $(-i, j)$.

(ii):

$$\begin{aligned} r(bd_N(F(-i,j))) &= r(bd_S(F(-i,j))) \\ &= r(bd_N(F(-i,j-1))) \\ &= bit_i(a(b \bmod 2^j)) \\ &= bit_i(a(b \bmod 2^{j+1})), \text{ since } b_j = 0. \end{aligned}$$

(iii): $l(bd_W(F(-i,j))) = 0$ for all blue tiles, which is correct because $b_j = 0$.

(2) ($b_j = 1$ and $cbit_i(a(b \bmod 2^j), a2^j) = 0$): only a yellow tile may attach in position $(-i, j)$ because $l(bd_W(F(-(i-1), j))) = 1$.

(ii):

$$\begin{aligned} r(bd_N(F(-i,j))) &= xor(r(bd_S(F(-i,j))), r(bd_E(F(-i,j)))) \\ &= xor(r(bd_N(F(-i,j-1))), r(bd_W(F(-(i-1),j)))) \\ &= xor(bit_i(a(b \bmod 2^j)), bit_{i-1}(a2^{j-1})) \\ &= xor(bit_i(a(b \bmod 2^j)), bit_i(a2^j)) \\ &= bit_i(a(b \bmod 2^{j+1})), \text{ because the incoming } cbit = 0. \end{aligned}$$

(iii): For all yellow tiles,

$$\begin{aligned} l(bd_W(F(-i,j))) &= 1 + and(r(bd_S(F(-i,j))), r(bd_E(F(-i,j)))) \\ &= 1 + and(r(bd_N(F(-i,j-1))), r(bd_W(F(-(i-1),j)))) \\ &= 1 + and(bit_i(a(b \bmod 2^j)), bit_{i-1}(a2^{j-1})) \\ &= 1 + and(bit_i(a(b \bmod 2^j)), bit_i(a2^j)) \\ &= 1 + cbit_{i+1}(a(b \bmod 2^j), a2^j). \end{aligned}$$

(3) ($b_j = 1$ and $cbit_i(a(b \bmod 2^j), a2^j) = 1$): only a green tile may attach in position $(-i, j)$ because $l(bd_W(F(-(i-1), j))) = 2$.

(ii):

$$\begin{aligned} r(bd_N(F(-i,j))) &= 1 - xor(r(bd_S(F(-i,j))), r(bd_E(F(-i,j)))) \\ &= 1 - xor(r(bd_N(F(-i,j-1))), r(bd_W(F(-(i-1),j)))) \\ &= 1 - xor(bit_i(a(b \bmod 2^j)), bit_{i-1}(a2^{j-1})) \\ &= 1 - xor(bit_i(a(b \bmod 2^j)), bit_i(a2^j)) \\ &= bit_i(a(b \bmod 2^{j+1})), \text{ because the incoming } cbit = 1. \end{aligned}$$

(iii): For all green tiles,

$$\begin{aligned} l(bd_W(F(-i,j))) &= 1 + or(r(bd_S(F(-i,j))), r(bd_E(F(-i,j)))) \\ &= 1 + or(r(bd_N(F(-i,j-1))), r(bd_W(F(-(i-1),j)))) \\ &= 1 + or(bit_i(a(b \bmod 2^j)), bit_{i-1}(a2^{j-1})) \\ &= 1 + or(bit_i(a(b \bmod 2^j)), bit_i(a2^j)) \\ &= 1 + cbit_i(a(b \bmod 2^j), a2^j). \end{aligned}$$

(iv): For all non-magenta tiles in position $(-i, j)$:

$$\begin{aligned} r(bd_W(F(-i,j))) &= l(bd_S(F(-i,j))) \\ &= l(bd_N(F(-i,j-1))) \\ &= bit_i(a2^{j-1}). \end{aligned}$$

Thus, for all $0 \le i < n_a + n_b$, $v(F(-i, n_b - 1)) = bit_i(a(b \bmod 2^{n_b})) = bit_i(ab)$. For all $j \in \mathbb{N}$, let $o_{s_1}(j) = (-j, -1)$, $o_{s_2}(j) = (1, j)$, and $o_f(j) = (-j, n_b)$. Because ab is no larger than $n_a + n_b$ bits, it follows that $\mathbb{S}_\times$ computes $f(a, b) = ab$. □

By the assembly time corollary (Corollary 2.2), the assembly time for this system is $\Theta(n_a + n_b)$. Note that $|\Gamma_\times| = 6$ and $|T_\times| = 28$.

3. Contributions

The tile assembly model is a formal model of self-assembly and crystal growth. I explored what it means for a system to compute a function. In other research studying such systems, two measurements arise as the important quantities: the number of tile types and the assembly speed of the computation. I adopted these measurements for their counterparts for systems that compute functions: the number of input tile types, the number of computation tile types, and the assembly speed of the computation. I explored these measurements for systems that add and multiply. I demonstrated an 8-computational-tile system for computing $f(a, b) = a + b$ and a 28-computational-tile system for computing $f(a, b) = ab$. Both systems use $\Theta(1)$ input tiles and compute in $\Theta(n)$ steps.

The theoretical analysis of systems that can assemble shapes or compute simpler functions [19,2,5,13] has led to experimental manifestations of such systems using DNA tiles [11,17]. While most DNA computation suffers from high error rates, self-assembly has yielded results in error correction [21,9,12,18] that may be used to decrease the error rates. Most experimental results in DNA computation have not appealed to the advantages of crystal growth; however, these early works on the fundamentals of self-assembly and the physical experimental evidence of actual DNA crystals suggest a bright future for DNA computation.

Acknowledgements

I wish to thank Dustin Reishus for his help formalizing the ideas presented in this paper.

References

[1] Gagan Aggarwal, Qi Cheng, Michael H. Goldwasser, Ming-Yang Kao, Pablo Moisset de Espanes, Robert T. Schweller, Complexities for generalized models of self-assembly, SIAM Journal on Computing 34 (6) (2005) 1493–1515.

[2] Leonard Adleman, Qi Cheng, Ashish Goel, Ming-Deh Huang, David Kempe, Pablo Moisset de Espanes, Paul Rothermund, Combinatorial optimization problems in self-assembly, in: ACM Symposium on Theory of Computing, STOC02, Montreal, Quebec, Canada, 2002, pp. 23–32.

[3] Leonard Adleman, Molecular computation of solutions to combinatorial problems, Science 266 (1994) 1021–1024.

[4] Leonard Adleman, Towards a mathematical theory of self-assembly, Technical Report 00-722, Department of Computer Science, University of Southern California, Los Angeles, CA, 2000.

[5] Leonard Adleman, Ashish Goel, Ming-Deh Huang, Pablo Moisset de Espanes, Running time and program size for self-assembled squares, in: ACM Symposium on Theory of Computing, STOC02, Montreal, Quebec, Canada, 2001, pp. 740–748.

[6] Leonard Adleman, Jarkko Kari, Lila Kari, Dustin Reishus, On the decidability of self-assembly of infinite ribbons, in: The 43rd Annual IEEE Symposium on Foundations of Computer Science, FOCS'02, Ottawa, Ontario, Canada, November 2002, pp. 530–537.

[7] Ravinderjit Braich, Nickolas Chelyapov, Cliff Johnson, Paul Rothemund, Leonard Adleman, Solution of a 20-variable 3-SAT problem on a DNA computer, Science 296 (5567) (2002) 499–502.

[8] Yuliy Baryshnikov, Ed G. Coffman, Petar Momcilovic, DNA-based Computation Times, in: Springer Lecture Notes in Computer Science, vol. 3384, 2005, pp. 14–23.

[9] Yuliy Baryshnikov, Ed G. Coffman, Nadrian Seeman, Teddy Yimwadsana, Self correcting self assembly: Growth models and the Hammersley process, in: Proceedings of the 11th International Meeting on DNA Computing, DNA 2005, London, Ontario, June 2005.

[10] Ravinderjit Braich, Cliff Johnson, Paul Rothemund, Darryl Hwang, Nickolas Chelyapov, Leonard Adleman, Solution of a satisfiability problem on a gel-based DNA computer, in: DNA Computing: 6th International Workshop on DNA-Based Computers, DNA2000, Leiden, The Netherlands, June 2000, p. 27.

[11] Robert Barish, Paul Rothemund, Erik Winfree, Two computational primitives for algorithmic self-assembly: Copying and counting, Nano Letters 5 (12) (2005) 2586–2592.

[12] Ho-Lin Chen, Ashish Goel, Error free self-assembly with error prone tiles, in: Proceedings of the 10th International Meeting on DNA Based Computers, DNA 2004, Milan, Italy, June 2004.

[13] Pablo Moisset de Espanes, Computerized exhaustive search for optimal self-assembly counters, in: The 2nd Annual Foundations of Nanoscience Conference, FNANO'05, Snowbird, UT, April 2005, pp. 24–25.

[14] Tsu Ju Fu, Nadrian C. Seeman, DNA double-crossover molecules, Biochemistry 32 (13) (1993) 3211–3220.

[15] Ming-Yang Kao, Robert Schweller, Reducing tile complexity for self-assembly through temperature programming, in: Proceedings of the 17th Annual ACM–SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, FL, January 2006, pp. 571–580.

[16] Raphael M. Robinson, Undecidability and nonperiodicity for tilings of the plane, Inventiones Mathematicae 12 (3) (1971) 177–209.

[17] Paul Rothemund, Nick Papadakis, Erik Winfree, Algorithmic self-assembly of DNA Sierpinski triangles, PLoS Biology 2 (12) (2004) e424.

[18] John H. Reif, Sadheer Sahu, Peng Yin, Compact error-resilient computational DNA tiling assemblies, in: Proceedings of the 10th International Meeting on DNA Based Computers, DNA 2004, Milan, Italy, June 2004.
[19] Paul Rothemund, Erik Winfree, The program-size complexity of self-assembled squares, in: ACM Symposium on Theory of Computing, STOC02, Montreal, Quebec, Canada, 2001, pp. 459–468.
[20] Hao Wang, Proving theorems by pattern recognition I, Bell System Technical Journal 40 (1961) 1–42.
[21] Erik Winfree, Renat Bekbolatov, Proofreading tile sets: Error correction for algorithmic self-assembly, in: The 43rd Annual IEEE Symposium on Foundations of Computer Science, FOCS'02, vol. 2943, Madison, WI, June 2003, pp. 126–144.
[22] Erik Winfree, On the computational power of DNA annealing and ligation, in: DNA Based Computers, 1996, pp. 199–221.
[23] Erik Winfree, Algorithmic self-assembly of DNA, Ph.D. Thesis, Caltech, Pasadena, CA, June 1998.