Universal Computation and Optimal Construction in the Chemical Reaction Network-Controlled Tile Assembly Model

Nicholas Schiefer and Erik Winfree(✉)

California Institute of Technology, Pasadena, CA 91125, USA
winfree@caltech.edu

Abstract. Tile-based self-assembly and chemical reaction networks provide two well-studied models of scalable DNA-based computation. Although tile self-assembly provides a powerful framework for describing Turing-universal self-assembling systems, assembly logic in tile self-assembly is localized, so that only the nearby environment can affect the process of self-assembly. We introduce a new model of tile-based self-assembly in which a well-mixed chemical reaction network interacts with self-assembling tiles to exert non-local control on the self-assembly process. Through simulation of multi-stack machines, we demonstrate that this new model is efficiently Turing-universal, even when restricted to unbounded space in only one spatial dimension. Using a natural notion of program complexity, we also show that this new model can produce many complex shapes with programs of lower complexity. Most notably, we show that arbitrary connected shapes can be produced by a program with complexity bounded by the Kolmogorov complexity of the shape, without the large scale factor that is required for the analogous result in the abstract tile assembly model. These results suggest that controlled self-assembly provides additional algorithmic power over tile-only self-assembly, and that non-local control enhances our ability to perform computation and algorithmically self-assemble structures from small input programs.

1 Introduction

Biological systems are capable of remarkable self-organization directed by information-carrying molecules and the complex biochemical networks that interpret them. Even more remarkably, these systems are able to modify themselves and reconfigure their structure in response to changes in the surrounding environment. In many areas of nanotechnology, we seek to emulate biological systems by implementing self-assembly processes and dynamical systems at the nanoscale.

Because of its relatively rigid, well-understood structure and the specificity of Watson-Crick hybridization, DNA is a common substrate for work in nanotechnology. So far, work in the field has been loosely divided into two classes: structural DNA nanotechnology, which involves self-assembly of small DNA subunits

A. Phillips and P. Yin (Eds.): DNA 2015, LNCS 9211, pp. 34–54, 2015.
DOI: 10.1007/978-3-319-21999-8_3

into larger structures [18,25], and dynamic DNA nanotechnology, which seeks to implement the behavior of dynamical systems through fluctuating quantities of chemical species [34]. Both classes of DNA nanotechnology have been explored extensively, now offering scalable methods for engineering complex nanoscale structures from small components [3,14,21,23,31] and both analog and digital circuits for a variety of tasks [7,20,24,32,35]. Furthermore, both classes have well-studied theoretical models, including a variety of self-assembly models based on Wang tiling [11,16] and models of abstract chemical reaction networks for chemical dynamics [6,8,27,28].

Despite the well-established results in these fields, little theoretical work has considered interactions between structural and dynamic DNA nanotechnology, suggesting that current theoretical models do not capture the full computational power of biomolecular systems. In biological systems, structure influences dynamics, and dynamics influences structure: the two are inextricably linked together. Recently, work by Zhang et al. [33] proposed and experimentally demonstrated a method for controlling the formation of DNA nanotubes from double-crossover tiles using an upstream catalytic circuit implemented as a DNA strand displacement system. However, there is currently little theoretical understanding of the computational power of interacting dynamic and structural biomolecular computing systems.

Some hints come from studies of the computational power of chemical systems involving linear polymers that can store information, which in theory can perform efficient and error-free Turing-universal computation [4,5,19]. In particular, a plausible theoretical implementation of Turing-universal stack machines using dynamic DNA nanotechnology showed that, at least for linear polymers, DNA strand displacement systems can control the assembly and disassembly of nanostructures in a very general and programmable way [19].

Here, we are interested in the ability of biomolecular systems to implement computation and construction tasks in two dimensions (or more): for the former, our goal is to perform a computation and report the answer, while in the latter, our goal is produce a particular nanostructure. Although tile self-assembly permits Turing-universal computation [22] and Kolmogorov-optimal construction (up to scale) [29], the assembly logic in tile self-assembly is local; only the immediate surroundings of a tile can influence its binding. In contrast, chemical reaction networks are usually formulated with a "well-mixed" assumption under which chemical species have no position within the reaction vessel. Although this allows highly non-local information transfer, it precludes the possibility of assembling large complexes. Consequently, we aim to leverage the non-local information transfer offered by chemical reaction networks to exercise non-local control over a two-dimensional (or, in principle, three-dimensional) tile assembly process.

Thus, we introduce the chemical reaction network-controlled tile assembly model (CRN-TAM), a formal model of molecular computing that uses chemical reaction networks to provide non-local control over a tile self-assembly process. In doing so, we formalize and generalize the type of biomolecular computing systems demonstrated experimentally by Zhang et al. [33] and explored theoretically by Qian et al. [19], allowing us to reason mathematically about the capabilities of such systems in comparison to other models of molecular programming. We show that the CRN-TAM subsumes models of stochastic chemical reaction networks and the abstract tile assembly model, and we establish a number of useful "building blocks" for CRN-TAM programs.

Through the Turing-universality of the aTAM, we demonstrate that the CRN-TAM is Turing-universal. Furthermore, we show that the CRN-TAM permits the efficient construction of multi-stack machines, proving that the CRN-TAM is also Turing-universal when restricted to unbounded space in only one spatial dimension, unlike other models of tile-based self-assembly.

Using a natural notion of program complexity, we then turn to bounding the complexity of a minimal CRN-TAM program that constructs a specified algorithmic shape. By explicit construction, we show that there is a CRN-TAM program that constructs every shape $\mathcal{S}$ at scale 2, with complexity bounded by the Kolmogorov complexity of $\mathcal{S}$. We show that this bound is tight by providing a matching lower bound.

2 Defining the CRN-TAM

We begin by outlining a formal definition of the chemical reaction network-controlled tile assembly model and providing a number of useful definitions.

Definition 1. *A* tile *is an oriented square with a bond on each side; the bond positions are called "north," "south," "east," and "west." Each* bond *has a distinct label and a strength, which is a non-negative integer. Formally, a bond is a tuple (ℓ, s) with label ℓ and strength $s \in \mathbb{N}$. For compactness, we often express a bond $(\ell, 1)$ evocatively as $-_\ell$ and a bond $(\ell, 2)$ as $=_\ell$. A tile is a four-tuple $\boxed{t} = (N, E, S, W)$ of bonds for the north, east, south, and west sides, respectively. Throughout this paper, tiles are denoted by symbols surrounded by boxes, as above.*

Definition 2. *An* assembly *is a function $A : \mathbb{Z}^2 \to (T \cup \{\varepsilon\})$ that gives the type of tile that occupies each site of the 2D lattice, where ε corresponds to an* empty *site. If $A(x, y) = \varepsilon$, then the site is said to be unoccupied, since there is no tile there. To be a valid assembly at temperature τ, A must satisfy these properties:*

- *The origin must be occupied by a tile $A(0, 0) \neq \varepsilon$, which we call the* seed *of the assembly.*
- *The occupied sites of the assembly must be connected.*
- *The total binding strength of each tile in the assembly is at least τ.*

Throughout this paper, assemblies are denoted by symbols surrounded by double boxes, e.g. $\boxed{\boxed{A}}$, or shown in a different color from tiles.

The definitions given in Definitions 1 and 2 are identical to those in previous models of tile-based self-assembly derived from the abstract tile assembly model [2,16,22,29]. Although it was implicit in previous "single-crystal" models such as the aTAM, here we explicitly distinguish between free tiles in solution and growing assemblies, even those that contain only a single tile. This ensures formally that free tiles only attach to "activated" assemblies, and not to each other, which is convenient for avoiding issues of spontaneous nucleation and essential for uniform treatment of the "removal signal" reactions described below. Further, it provides a natural way for our model to allow for multiple crystals growing within the same system.

In tile self-assembly, a molecular program is specified by a set T of tiles and their associated bond strengths, an initial seed tile, and a temperature. Analogously, we can define the structural form of a CRN-TAM program:

Definition 3. *A* program *under the chemical reaction network-controlled tile assembly model is a tuple* (S, T, R, τ, I) *where*

- *S is a finite set of identified signal species.*
- *T is a finite set of tuples $\left(\boxed{t}, t^*\right)$, where $\boxed{t}$ is a tile and t^* is either ε or some signal species in S. The species t^*, if it exists, is called the* removal signal *for tile $\boxed{t}$. No tile may appear in more than one tuple.*
- *R is a set of reactions, each of the form:*
  - *$A+B \xrightarrow{k} C+D$ for signals $A, B, C, D \in \{\varepsilon\} \cup S$. These are the "normal" CRN reactions.*
  - *$A + \boxed{T} \xrightarrow{k} C + D$ for signals $A, C, D \in \{\varepsilon\} \cup S$ and tile $\boxed{T}$. These are tile deletion reactions.*
  - *$A + B \xrightarrow{k} \boxed{T} + C$ or $A + B \xrightarrow{k} \boxed{T} + \boxed{T'}$ for signals $A, B, C \in \{\varepsilon\} \cup S$ and tiles $\boxed{T}$ and $\boxed{T'}$. These are tile creation reactions.*
  - *$A + \boxed{T} \xrightarrow{k} B + \boxed{T'}$ for signals $A, B \in \{\varepsilon\} \cup S$ and tiles $\boxed{T}$ and $\boxed{T'}$. These are tile relabelling reactions.*
  - *$A + \boxed{X} \xrightarrow{k} \boxed{\boxed{X}} + X^*$, where $A \in \{\varepsilon\} \cup S$ and $\left(\boxed{X}, X^*\right) \in T$. This tile activation reaction converts a free tile into the seed of a new assembly.*
  - *$\boxed{\boxed{X}} + X^* \xrightarrow{k} A + \boxed{X}$, where $A \in \{\varepsilon\} \cup S$ and $\left(\boxed{X}, X^*\right) \in T$. This tile deactivation reaction converts a seed tile assembly into a free tile.*

  In all of these reactions, k is some rate constant. All of the constructions in this paper are independent of rate constant, and so it is often omitted for notational simplicity. In all cases where a rate constant is omitted, it can be assumed to be 1. When any reactant or product is taken to be ε, the interpretation is that the reactant or product does not exist; for example, a reaction $A + \varepsilon \xrightarrow{k} \varepsilon + D$ is just $A \xrightarrow{k} D$.
- *$\tau \in \mathbb{N}$ is the temperature, or minimum binding strength, typically 0, 1, or 2.*
- *I is an initial state, which is a multiset of tiles and signals that are initially present. Often we will treat I as a function $I : (S \cup T) \to \mathbb{N}$ where $I(z)$ is the count of species z in the multiset. No assemblies are initially present.*

The elements of the set S of signal species are analogous to "normal" species in a chemical reaction network and the set T of tiles is analogous to an aTAM tile

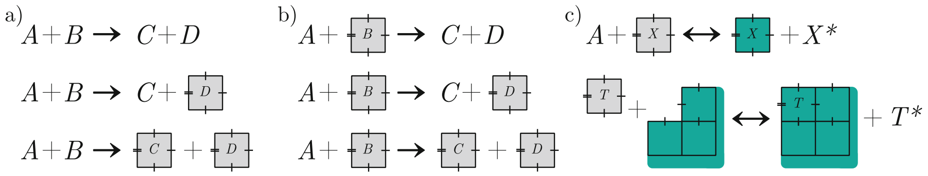

Fig. 1. Example reactions for a CRN-TAM program. (a) Normal chemical reaction network reactions and tile creation reactions. (b) Tile deletion and relabelling reactions. (c) Tile activation, deactivation, addition, and removal reactions.

set (except in that tile concentrations are held constant in the aTAM, while in the CRN-TAM discrete counts of tiles are tracked and may change as reactions proceed). As in the aTAM, tiles may interact with assemblies to form larger structures. However, in the CRN-TAM, each assembly step is accompanied by the release of the tile's associated removal species, and the reaction may be reversible if the removal species is not ε. As a result, the behavior of a CRN-TAM program will be dictated not only by the explicitly specified reactions R as above, but also the *tile addition and removal reactions*:

Definition 4. *A tile addition reaction has the form*

$$\boxed{\boxed{\alpha}} + \boxed{t} \xrightarrow{1} \boxed{\boxed{\beta}} + t^*$$

wherever α and β are valid assemblies that differ by exactly one tile, $\boxed{t}$, that is in β but not in α, where the tuple $\left(\boxed{t}, t^*\right) \in T$. Since β is valid, $\boxed{t}$ formed new bonds with total strength at least τ. The corresponding removal reaction

$$\boxed{\boxed{\beta}} + t^* \xrightarrow{1} \boxed{\boxed{\alpha}} + \boxed{t}$$

may occur only when $\boxed{t}$ is bound by exactly *strength τ.*

We add the condition that a tile removal can only occur if the tile is bound with strength exactly τ based on the principle that reversible reactions should be roughly energetically balanced. As a side effect, it prevents a removal signal from "ripping out" a tile from the middle of an assembly, and thus enforces that only tiles at the boundary can be removed, which corresponds naturally with tile removal in the kinetic tile assembly model.

Together, the contents of a reaction vessel—including free species and assemblies—completely specify the state of a CRN-TAM program at any point: A non-exhaustive sample of allowed reaction types is illustrated in Fig. 1.

Definition 5. *A* state *L of a CRN-TAM program P is a multiset of signals, tiles, and assemblies.*

Definition 6. *The* propensity *of a reaction is the product of its rate constant and the count of each of its reactants. The* possible reactions *of a state L of a CRN-TAM program $P = (S, T, R, \tau, I)$ are all of the reactions in R with non-zero propensity and all tile addition or removal reactions with non-zero propensity.*

Over time, the program evolves from the initial state according to stochastic Gillespie dynamics: that is, reactions occur at a rate proportional to their current propensity [13]. The time evolution of the state of a CRN-TAM program therefore forms a continuous-time Markov chain.

Definition 7. *An assembly is* terminal *with respect to a state if there can be no possible tile addition or removal reactions involving that assembly in the future.*

While proving that an assembly is terminal can occasionally be done by examination of just the assembly itself (showing that there is no location where a tile may be added or removed, whether or not the tile or removal signal exists in solution), showing that an assembly is terminal is generally undecidable.

Definition 8. *Although the time evolution of the state of a CRN-TAM system evolves stochastically, we may speak of* deterministic *CRN-TAM systems: systems for which there is at most one possible forward reaction, and at most one possible reverse reaction, for every state. That is, the system state space is a one-dimensional line.*

Definition 9. *A CRN-TAM program acting on an initial state L* stops *if the set of reachable states is finite and reaches a state with no possible further reactions with probability one.*

Definition 10. *A CRN-TAM program* constructs *a shape $\mathcal{S}$ if the program stops with precisely one terminal assembly, and that assembly has shape $\mathcal{S}$.*

To give a natural notion of the "size" of a CRN-TAM program, we introduce a notion of program complexity. This notion is analogous to the tile set size under the aTAM, or the number of signals and reactions in a CRN.

Definition 11. *The complexity of an initial state $I : (S \cup T) \to \mathbb{N}$ is*

$$|I| = \sum_{z \in (S \cup T)} \log_2(I(z) + 1)$$

This definition is natural since it is the number of bits needed to specify a general initial state I, up to small constant multiplicative and additive factors.

Definition 12. *Let $P = (S, T, R, \tau, I)$ be a CRN-TAM program (with unit reaction rate constants). The* complexity *of P with respect to temperature τ is*

$$K_{\mathrm{CT}}^{\tau}(P) = |S| + |T| + |R| + |I| = |S| + |T| + |R| + \sum_{z \in (S \cup T)} \log_2(I(z) + 1)$$

Each of the terms is related, up to logarithmic factors, to the amount of information needed to specify that component of a CRN-TAM program.

3 Preliminary Results

The CRN-TAM is based on its eponymous models: abstract chemical reaction networks and the abstract tile assembly model. As one would hope, it subsumes both of these models. Subsuming models of stochastic CRNs is trivial:

Theorem 1. *For any chemical reaction network C with species S and reactions R (each of which has at most two reactants and two products), there is a CRN-TAM program $P = (S, \varnothing, R, 0, L)$ with dynamics identical to those of C acting on L.*

In contrast, the abstract tile assembly model requires an unbounded supply of each tile type. Thankfully, it is straightforward to generate this supply with a CRN-TAM program:

Theorem 2. *Let T be a set of tiles for the abstract tile assembly model at temperature τ, and suppose that $\boxed{T_0} \in T$ is the designated seed tile. There is a CRN-TAM program P that simulates the operation of T, in terms of reachable assemblies, with complexity $K^{\tau}_{\mathrm{CT}}(P) \in \Theta(|T|)$.*

Proof. For each tile $\boxed{t} \in T$, we introduce the species C_t and the catalytic reaction $C_t \to C_t + \boxed{t}$. The removal signal of every tile is ε to enforce irreversibility of tile addition. Our initial state consists of one of each C_t and the seed tile $\boxed{T_0}$. The reaction $\boxed{T_0} \to \boxed{\boxed{T_0}}$ initiates the assembly process. □

Next, we introduce a number of basic constructions that demonstrate the flavor of CRN-TAM programs. The most important of these gives an efficient way to run a broad-class of CRN-TAM programs a certain number of times.

Definition 13. *A CRN-TAM program C is a* handshake subroutine *with respect to a set of data molecules D if it satisfies:*

Data-Inertness Property: *In any state consisting only of molecules in D, no reaction may occur.*

Single-Entry Property: *There is a species S that initiates the operation of C. That is, no reaction will take place until a single molecule of species S appears, and that molecule is consumed in the first reaction of C.*

Single-Exit Property: *There is a species F that signifies the completion of C's operation; we say that F terminates C. That is, F does not appear while reactions of C are still possible, and F is produced by the last reaction of C that can happen.*

Intuitively, handshake subroutines are programs that we can choose to start and can know have stopped. The definition does not require that C always stops, but in typical usage there will be an argument that it does.

Lemma 1. *Let k be a nonnegative integer and P be a handshake subroutine that is initiated by species P_{S} and terminated by species P_{F}. There exists a handshake subroutine* powerCounter(k, A, B, P)*, initiated by A and terminated by B, with $\Theta(k)$ additional chemical species and $\Theta(k)$ additional chemical reactions that runs P exactly $2^k - 1$ times.*

Proof. Let $\#(a)$ be the number of molecules of species a at a specified time.

We introduce the sequences of species $X_0, X_1, \ldots, X_{k-1}$, $S_1, \ldots, S_k$, and $Y_0, Y_1, \ldots, Y_{k-1}$, all of which do not appear in P (i.e. are "new" species). We construct our counting circuit as a binary counter, where the pair (X_i, Y_i) is a dual-rail representation of the state of the ith bit of the counter; that is, if $\#(X_i) = 1$, then $\#(Y_i) = 0$, and if $\#(X_i) = 0$, then $\#(Y_i) = 1$. By convention, the value of $\#(X_i)$ is the value of the bit. The S_i species will serve as (single-rail) digit carry markers. For k-bit counting, we produce a new program as follows:

1. For each bit i, we introduce the reactions

$$S_i + Y_i \rightarrow X_i + P_{\mathrm{S}}, \quad S_i + X_i \rightarrow Y_i + S_{i+1}$$

2. Add the reaction $P_{\mathrm{F}} \rightarrow S_0$, to continue counting after C runs once.
3. Add the reaction $A \rightarrow P_{\mathrm{S}}$ to start the binary counting.
4. Add the reaction $S_k \rightarrow B$ to indicate that the counting is finished.

The initial state of our binary counting program is the full collection of species $Y_0, \ldots, Y_{k-1}$, one copy each, indicating that the counter starts at 0.

Notice that the structure of the reactions ensures that the following properties hold by simple induction:

- At every time between the consumption of A and the creation of B, there is exactly one of the S_i carry species at all times, since every other reaction consumes one of these and produces one of these.
- At any time, exactly one of $\{X_i, Y_i\}$ is present, since every reaction "flips a bit" by consuming one $\{X_i, Y_i\}$ and producing the other.
- The reactions implement precisely the carry behavior of a binary counter with k bits.
- At any time, there is only one reaction that can take place, by the above properties, and so the program works deterministically.
- The initiation signal P_{S} is released and consumed precisely once for every one of the $2^k - 1$ values that the counter's species' can encode.
- Between successive releases of S_0, the program P is run exactly once.

Observe that in our constructed k-bit counter, we introduce $\Theta(k)$ species and $\Theta(k)$ reactions beyond those of the original program. Thus, the program—which we call powerCounter(k, A, B, P)—has the desired properties. □

We can easily modify this construction to run a handshake subroutine exactly $n \in \mathbb{N}$ times, using $\Theta(\log n)$ signals and reactions:

Theorem 3. *Let n be a positive integer and P be a handshake subroutine that is initiated by species P_{S} and terminated by species P_{F}. There exists a handshake subroutine* binaryCounter(n, A, B, P)*, initiated by A and terminated by B, with $\Theta(\log n)$ additional species and $\Theta(\log n)$ additional bimolecular reactions that runs P exactly n times.*

Proof. By Lemma 1, we can construct a handshake subroutine that runs P a total of $2^{\lceil \log n \rceil} - 1$ times using $\Theta(\lceil \log n \rceil) = \Theta(\log n)$ additional species and reactions. Extending this to run P one more time using the reactions $S_{\lceil \log n \rceil} \rightarrow P_{\mathrm{S}}$ and $P_{\mathrm{F}} \rightarrow B$ instead of $S_{\lceil \log n \rceil} \rightarrow B$, we have a handshake subroutine that runs P a total of $2^{\lceil \log n \rceil}$ times.

We further modify our construction from Lemma 1 by adding a different initial state. Let $b_0 b_1 \cdots b_{\lceil \log n \rceil}$ be the unique binary representation of $2^{\lceil \log n \rceil} - n \geq 0$, by definition. Then, our initial state (instead of a full sequence of "off" bits Y_i) will be, for all i, $\#(X_i) = b_i$ and $\#(Y_i) = 1 - b_i$.

In effect, our $\lceil \log n \rceil$-bit counter starts with a value of $2^{\lceil \log n \rceil} - n$, which we identify as "zero", and ends in the state $2^{\lceil \log n \rceil}$, which is therefore identified as $2^{\lceil \log n \rceil} - 2^{\lceil \log n \rceil} + n = n$. This construction adds only $\Theta(1)$ complexity beyond that from Lemma 1 so our program still has $\Theta(\log n)$ additional complexity. $\square$

It is intuitively useful to consider the initial state I of a CRN-TAM program P. However, the following theorem shows that the addition of the initial state does not provide extra *algorithmic power* over the model with no additional state.

Theorem 4. *Let $P = (S, T, R, \tau, I)$ be any CRN-TAM program that cannot start until some species F is released. We define a special signal Q^* and let our initial state $I' = \{Q^*\}$. There exists a program $P' = (S', T', R', \tau, I')$ with $K^{\tau}_{\mathrm{CT}}(P') \in \Theta(K^{\tau}_{\mathrm{CT}}(P))$ that has the same graph of possible states after $\Theta(|I|)$ initial states.*

Proof. Let $Z \subseteq (S \cup T)$ be the set of $s \in (S \cup T)$ with $I(s) > 0$, and let $\tilde{Z} = (z_1, z_2, \ldots, z_{|Z|})$ be an arbitrary ordered sequence of Z.

For each z_i, let C_i be the chemical reaction network $Q_i \rightarrow z_i + H_i$, and note that C is a handshake subroutine that simply creates one z_i. We construct P' by augmenting P with signal $Q_{|Z|+1}$, reactions $H_i \rightarrow Q_{i+1}$, $Q^* \rightarrow Q_1$, and $Q_{|Z|+1} \rightarrow F$, and for each z_i, a new signal Q_i and a CRN binaryCounter$(I(z_i), Q_i, H_i, C_i)$ that uses new species each time for its internal operation.

This construction is illustrated in Fig. 2. By Theorem 3, each binary counter will produce precisely $I(z_i)$ of each species z_i when Q_i is present.

We can now show that when Q_{i+1} is released, we have the correct "initial state" counts of all species $z_j, j \leq i$. In the base case, notice that the presence of Q^* caused the release of Q_1, which will cause the release of $I(z_1)$ of species z_1. Now, suppose that when Q_{k+1} is released, we have the correct counts of all species $z_j, j \leq k$. Then, notice that the release of Q_{k+1} initiates the release of exactly $I(z_{k+1})$ of z_{k+1}, and also the release of Q_{k+2}. By induction, when species $Q_{|Z|+1}$ is released, all of the species $z_i \in Z$ will have the correct initial counts.

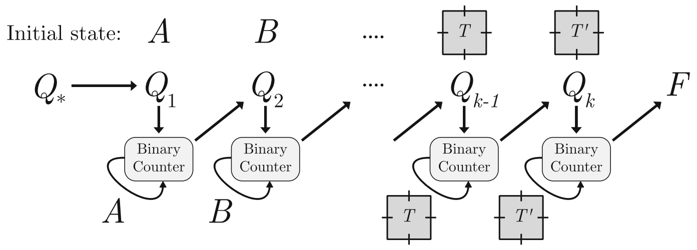

Fig. 2. Conceptual illustration of the construction for Theorem 4. Starting with the singleton "starter species", we use the binary-counting CRNs from Theorem 3 to produce the correct number of each initial species (as indicated above each stage). When each binary counter finishes, it starts the binary counter for the next species.

Since C_i requires a constant number of species and reactions, by Theorem 3, each binary counter and the associated reactions contribute $\Theta(\log I(z_i))$ complexity. Thus, by construction, P' has complexity:

$$K^{\tau}_{\mathrm{CT}}(P') = |S| + |\{Q_i\}^k_{i=1}| + |T| + |R| + \Theta(1) + \sum_{j=1}^{k} \Theta(\log I(z_j))$$

$$= |S| + |T| + |R| + \sum_{j=1}^{k} \Theta(\log I(z_j)) = \Theta(K^{\tau}_{\mathrm{CT}}(P))$$

□

So long as our CRN-TAM program is deterministic at its starting state, the initial state does not enable asymptotic reduction in program complexity. A careful reader will notice that the notion of program equivalence introduced in Theorem 4 is a restriction of the notion of weak bisimulation, which rigorously establishes a notion of equivalence for concurrent systems. In this paper, all of our constructions use deterministic CRN-TAM programs, and so this theorem will always apply.

This theorem also has a very simple corollary that immediately tells us that the CRN-TAM is in some ways more powerful than the aTAM:

Corollary 1. *A $1 \times n$ rectangle can be constructed by a program P with only a singleton initial state with $K^{\tau}_{\mathrm{CT}}(P) = \mathcal{O}(\log n)$.*

Proof. Let $P = \left(\{S\}, \left\{\left(\boxed{x}, \varepsilon\right)\right\}, \left\{S + \boxed{x} \rightarrow \boxed{\boxed{x}}\right\}, 1, I\right)$ where $I\left(\boxed{x}\right) = n$, $I(S) = 1$, and $\boxed{x}$ is simply a tile with the same strength-1 glue on two opposite sides (say, east and west). Clearly, P assembles a $1 \times n$ rectangle. By definition, $K^{\tau}_{\mathrm{CT}}(P) = \mathcal{O}(\log n)$, and note that this program will not start building until a seed tile is release. By Theorem 4, there exists a program P' that assembles the $1 \times n$ rectangle using only a singleton initial state with $K^{\tau}_{\mathrm{CT}}(P') = \mathcal{O}(\log n)$. □

In contrast, the size of a tile set (the notion of program complexity for the aTAM) that produces a $1 \times n$ rectangle is $\Theta(n)$ [2]. Similar bounds have been established for other models of tile self-assembly, including the negative glues model [17]. In this example, the lower complexity achieved by the CRN-TAM comes from the explicit control over the number of tiles of a particular type that are present in solution. One can imagine a formulation of the aTAM where similar exact counts of tiles are tracked as they are consumed; in such a model, a program with an initial state consisting of exactly n copies of tile x will construct a $1 \times n$ rectangle with $\mathcal{O}(1)$ tile types. However, Theorem 4 shows that the CRN-TAM can generate its initial state without changing the program complexity, while the analogous result for the modified aTAM would not hold (if program complexity is still taken to be just the number of tile types).

The CRN-TAM also permits the construction of exactly m copies of a shape that can be constructed deterministically:

Theorem 5. *Given a deterministic CRN-TAM program P at temperature τ that constructs a shape $\mathcal{S}$, there is a CRN-TAM program P' that constructs m copies of $\mathcal{S}$ with complexity $K^{\tau}_{\mathrm{CT}}(P') = \mathcal{O}(K^{\tau}_{\mathrm{CT}}(P) + \log m)$.*

Proof (sketch). Since P constructs $\mathcal{S}$ deterministically, at each time there is at most one possible tile addition. Furthermore, the tile addition must be done with a handshake (e.g. $X \to \boxed{t} + W$, $W + t^* \to Y$) because otherwise whether the assembly step or the next reaction occurs first would be non-deterministic. We construct P' by replacing the release and handshake of a single tile by P with a binary counter that, m times, releases the tile and waits for it to attach before continuing. That is, we invoke binaryCounter$(m, X, Y, \{S \to \boxed{t} + W, W + t^* \to F\})$. Note that the first release tile must be able to attach in a unique location on the m identical assemblies, because otherwise P would not have been deterministic. To ensure that each subsequent released tile attaches to a distinct assembly, rather than two or more of them attaching to each other on the same assembly, we label all of the tiles in P with the color red, and introduce an identical set of tiles with color black. We adjust the bonds so that red tiles may only bond to black tiles, and black tiles may only bond to red tiles. At each step, depending on the color of the tile we are trying to bind to, we release a tile of the appropriate color. This creates a checkerboard, and ensures proper assembly. Since P is deterministic, we only need a constant number of binary counters, one for each tile type. By Theorem 3, this takes $\mathcal{O}(\log m)$ extra complexity. Similarly, each step of P that creates a new seed assembly must instead create m seed assemblies. □

In defining the model, we introduced the condition that a removal reaction may only occur when the corresponding tile is bound to the rest of the assembly with strength exactly the temperature τ. The next theorem shows that this prevents assemblies from "falling apart" once they have been constructed, except in an order that is approximately the reverse of the order of addition.

Definition 14. *A site (i, j) containing tile $\boxed{x}$ is* dependent *on site (i', j') containing tile $\boxed{x'}$ if $\boxed{x}$ was added to the assembly after $\boxed{x'}$ and either shares a bond with $\boxed{x'}$ or shares a bond with a tile that is dependent on $\boxed{x'}$.*

This recursive definition of dependency imposes an implicit directed, acyclic graph of dependencies. To disassemble and assembly, we must recursively remove tiles from the leaves of the dependency DAG:

Theorem 6. *If Q is the set of all sites that are dependent on a site (a, b), then for any site $(i, j) \in Q$, the tile at site (i, j) cannot be removed until the tiles at all other sites in Q have been removed. That is, a tile at a site cannot be removed until the tiles at all dependent sites have been removed.*

One (limiting) consequence of Theorem 6 is the impossibility of creating "temporary scaffolding": a CRN-TAM program cannot build permanent parts of an assembly that are dependent on parts that are to be removed later, or it will be impossible to remove the scaffolding.

4 Turing-Universality

With these preliminaries, we consider the ability of CRN-TAM programs to simulate the operation of a Turing machine—which allows it to perform arbitrary computation—under various circumstances.

Theorem 7. *The CRN-TAM is Turing-universal at temperature $\tau = 2$.*

Proof. Given an aTAM tile set T, we may construct a CRN-TAM program $P = (\varnothing, T, \varnothing, 2, I_{\text{seed}})$ that simulates it at temperature 2. Thus, the CRN-TAM is Turing-universal by the Turing-universality of the aTAM [22]. □

Since the CRN-TAM subsumes the aTAM, Theorem 7 is far from surprising. However, the construction used to show the Turing-universality of the aTAM relies critically on the ability for Wang tilings to represent computation histories: the state of the Turing machine tape at each step. Importantly, this requires potentially unbounded space in both spatial dimensions. Through clever use of DNA strand displacement polymers, Qian et al. [19] showed the Turing universality of polymer reaction networks by constructing multi-stack machines. Since each stack is one-dimensional, the construction requires unbounded space in only one spatial dimension to provide Turing universality. A related construction for the CRN-TAM provides an analogous result:

Lemma 2. *Consider a (deterministic or non-deterministic) stack machine $M = (Q, \Sigma, \delta, n, q_0)$, consisting of a finite set of states Q, a symbol alphabet Σ, an integer n giving the number of stacks, an initial state $q_0 \in Q$, and a set of transition rules δ where each element of δ is one of:*

1. *$\alpha_1 \rightarrow \alpha_2$ for states $\alpha_1, \alpha_2 \in Q$.*
2. *$\alpha_1 \xrightarrow{\texttt{pop}_j = \sigma} \alpha_2, \sigma \in \Sigma$ for states $\alpha_1, \alpha_2 \in Q$, corresponding to popping a symbol off of stack j.*
3. *$\alpha_1 \xrightarrow{\texttt{push}_j(\sigma)} \alpha_2$ for states $\alpha_1, \alpha_2 \in Q$ and symbol $\sigma \in \Sigma$, corresponding to pushing a symbol σ onto stack j.*

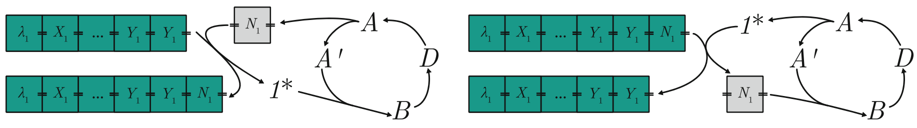

Fig. 3. Conceptual illustrations of the push and pop operations for a stack machine. *On the left*, the push operation "state A goes to state B, pushing symbol N onto stack 1" is shown. During a push operation, a stack-specific symbol is released, along with an intermediate species A'. The finite control waits for the stack- (but not symbol-) specific removal signal 1^* before continuing. *On the right*, the pop operation "from state A, pop a symbol from stack 1. If it is an N, go to state B" is shown. The pop operation is basically the reverse of the push operation. In both diagrams, D represents a network of other reactions, not just a single signal.

There is a CRN-TAM program $P = (S, T, R, 1, I)$ that simulates M with $K^1_{\mathrm{CT}}(P) = \mathcal{O}(|Q| + |\Sigma| + |\delta|)$. Furthermore, it requires unbounded space in only one geometric dimension, and runs in constant space in the other.

Proof. We show the result by construction, showing how to "compile" a stack machine M into a CRN-TAM program P. Our construction, based on the one used by [19] and illustrated in Fig. 3, is:

– For each state $\alpha_i \in Q$, we introduce the species S_i and S_i', including the initial state q_0 (represented by S_0).
– For each stack k, we introduce:
  - A "stack query species" Q_k
  - For each symbol $\sigma \in \Sigma$, a tile $\boxed{\sigma_k} = (\varnothing, -_k, \varnothing, -_k)$, each with the same removal signal Q_k
  - A tile $\boxed{\lambda_k} = (\varnothing, -_k, \varnothing, \varnothing)$ that represents the "bottom-of-stack", with removal signal ε.
– For each transition in δ:
  1. Implement transitions of the form $\alpha_i \to \alpha_j$ with the reaction
$$S_i \to S_j$$
  2. Implement transitions of the form $\alpha_i \xrightarrow{\mathtt{pop}_\ell = \sigma} \alpha_j$ with the reactions
$$S_i \to Q_\ell + S_i', \quad \boxed{\sigma_\ell} + S_i' \to S_j$$
  3. Implement transitions of the form $\alpha_i \xrightarrow{\mathtt{push}_\ell(\sigma)} \alpha_j$ with the reactions
$$S_i \to \boxed{\sigma_\ell} + S_i', \quad S_i' + Q_\ell \to S_j$$
– Include as initial state one of each $\boxed{\lambda_k}$ and one of species S_0. Include the reactions $\boxed{\lambda_k} \to \boxed{\boxed{\lambda_k}}$ to produce the initial assemblies.

The idea behind this construction is to represent the finite state with the presence of one of the S_i signals, and use tiles to construct a physical "stack" assembly of tiles for each stack in the stack machine. By identifying each tile with a stack's identity, we are able to restrict the symbol to interactions with the desired stack. To show correctness, we demonstrate that the transitions are correct, and argue that the possible reactions at each point are precisely the valid transitions from the current state in δ.

Clearly, reactions of type (1) are correct, since the reaction $S_i \rightarrow S_j$ is a direct implementation of the transition $\alpha_i \rightarrow \alpha_j$.

Reactions that perform pushes and pops are somewhat more complicated. To pop a symbol from stack ℓ, we produce the stack query species Q_ℓ and the "state storage species" S_i' that indicates that we are in the process of transitioning out of state α_i. Since Q_ℓ is the removal species of every one of the $\boxed{\sigma_\ell}, \forall \sigma \in \Sigma$, this will remove the tile at the only "exposed" site of stack ℓ. Then, this tile $\boxed{\sigma_\ell}$ can react with S_i' to produce the state transition to S_j.

Similarly, to push symbol σ onto stack ℓ, we release $\boxed{\sigma_\ell}$ with the reaction $S_i \rightarrow \boxed{\sigma_\ell} + S_i'$, releasing S_i' to indicate that we are in the process of transitioning out of state α_i. When the tile $\boxed{\sigma_\ell}$ attaches to the assembly for stack ℓ, it will release its removal species Q_ℓ. The removal species will then react with S_i', which transitions to another state.

Lastly, note that at any point in time, exactly one of the S_i or S_i' is present, and every reaction both produces and consumes exactly one S_i or S_i'. Thus, the progress of the CRN is deterministic, except possibly where a single S_i could transition in several ways (if the original stack machine is non-deterministic). We introduce a constant number of species and reaction to represent each state, stack symbol, and transition rule, and so $K^1_{\mathrm{CT}}(P) = \mathcal{O}(|Q| + |\Sigma| + |\delta|)$. □

Theorem 8. *Let U be a Turing machine. There exists a CRN-TAM program that simulates the operation of U, and requires unbounded space in only one geometric dimension, while running in constant space in the other. That is, the CRN-TAM is Turing-universal when running in one spatial dimension.*

Proof. It is well known that multi-stack pushdown automata are equivalent to Turing machines [26]. Observe that a multi-stack pushdown automaton can be implemented using the same construction used in Lemma 2. Thus, U may be implemented by way of an equivalent multi-stack pushdown automaton. □

There are two critical differences between Theorems 7 and 8. First, the aTAM is Turing-universal only at temperature 2, since algorithmic self-assembly is required for universality. In contrast, the CRN-TAM can simulate stack machines at temperature 1, and is thus Turing-universal at all nonzero temperatures. Although this is believed to be impossible for the temperature 1 aTAM, a 3D generalization aTAM is Turing-universal at temperature 1; furthermore, the construction requires only constant space in the third dimension [9].

Second, the CRN-TAM supports Turing-universal computation using unbounded space in only one spatial dimension, while the aTAM requires unbounded space in both spatial dimensions for universality. Interestingly, negative glues also allow for a restricted version of this result, where assemblies can be split into one-dimensional assemblies but cannot be deconstructed completely [12].

5 Optimal Encoding of Binary Strings

Having given efficient constructions for simulating Turing-universal computation, we now consider the related problem of efficiently encoding inputs for these Turing machines as (binary) strings. Given our stack machine construction, we aim to *encode* strings of length n as $1 \times n$ tile assemblies, which can be used as the input for a stack machine like in Lemma 2.

Definition 15. *A CRN-TAM program P* encodes *a binary string x if it constructs a $1 \times |x|$ rectangular assembly of tiles representing the bits of x.*

Of course, a binary string x of length n can be easily encoded by a CRN-TAM program of $\Theta(n)$ unique tile types: one for each bit of x. However, just as Adleman et al. [1] and Soloveichik and Winfree [29] encoded strings of length n in smaller aTAM tile sets that self-assemble at temperature $\tau = 2$ to unpack the bits, in the CRN-TAM we can do substantially better at $\tau = 1$ and using just one dimension for self-assembly:

Theorem 9. *For any binary string x of length n, there is a CRN-TAM program $P = (S, T, R, 1, I)$ that encodes x and has complexity $K^1_{\mathrm{CT}}(P) = \mathcal{O}(n/\log n)$.*

Proof. Suppose that we represent x as a sequence of k binary words $(w_1, w_2, \ldots, w_k)$, each with size $w = n/k$. For convenience, define the functions $h(x)$ and $t(x)$ to be the head and tail of a binary string x (i.e. the first bit, and all the remaining bits, respectively).

Define the data tiles $\boxed{T_0} = (\varnothing, -, \varnothing, -)$ and $\boxed{T_1} = (\varnothing, -, \varnothing, -)$, encoding the binary symbols zero and one, with removal signals 0^* and 1^*, respectively. Additionally, define the "bottom-of-stack" tile $\boxed{\lambda} = (\varnothing, -, \varnothing, \varnothing)$ that will create the seed assembly through an activation reaction.

For every binary string a of length at most w, we introduce several signals: A_a (the "construction signal"), A'_a (the "intermediate signal"), W_a (the "wait signal"), and B_a (the "completion signal"). For any binary string a of length at most w, define the following set R_a of reactions:

$$R_a = \{A_a \to A'_a + \boxed{T_{h(w)}},\ h(w)^* + A'_a \to W_a + A_{t(w)},\ B_{t(w)} + W_a \to B_w\}$$

By construction, the reactions in R_a push the first bit of a onto the stack, then invoke the reaction gadget for the remaining bits of a. It waits to receive the completion signal for the tail bits of a, and then issues its own completion signal. Notice that R_a has constant size.

To encode x, we can use k hard-coded tiles of distinct tile types that will assemble together; each tile represents a w-bit word of x; equivalently, we could use a hard-coded CRN producing unique signals for each word. We can then include the tiles, signals, and reactions described above for every string of length at most w, along with reactions that will repeatedly pop a word-tile representing word a from the stack of word tiles, produce A_a, and wait to consume B_a before popping the next word tile. These require only a constant number of reactions, so the total program complexity is: $K^1_{\mathrm{CT}}(P) = \Theta(k) + \mathcal{O}(2^{w+1})$ since there are 2^{w+1} strings of length at most w.

Picking $w = \log(n/\log n)$, we get $k = n/(\log n - \log\log n) \in \mathcal{O}(n/\log n)$, and so $K^\tau_{\mathrm{CT}}(P) = \mathcal{O}(n/\log n)$. □

6 Kolmogorov-Optimal Assembly of Algorithmic Shapes

We now turn our attention to the problem of constructing a geometric shape $\mathcal{S}$ using the CRN-TAM program of minimal complexity. Following [29], we can give a formal definition of shape:

Definition 16. *A* shape $\mathcal{S}$ *is a connected subset of the 2-dimensional lattice* $\mathbb{Z}^2$ *under the equivalence relation* $\mathcal{S} = \mathcal{S}'$ *if and only if* $\mathcal{S}'$ *is a translation of* $\mathcal{S}$*. A* c-scaling *of a shape* $\mathcal{S}$ *is the shape* $s_c(\mathcal{S})$ *that is obtained by replacing each square of* $\mathcal{S}$ *with a* $c \times c$ *block of squares.*

We follow the usual definition of the *Kolmogorov complexity* of a binary string x with respect to a fixed universal Turing machine U as the minimal size of a program for U that outputs x. We extend this notion to a shape $\mathcal{S}$:

Definition 17. *The* Kolmogorov complexity *of a shape* $\mathcal{S}$ *with respect to a universal Turing machine* U *is the minimal size of a program for* U *that outputs* $\mathcal{S}$ *as a list of coordinates. We denote the Kolmogorov complexity of* $\mathcal{S}$ *by* $K(\mathcal{S})$*.*

Definition 18. *The* CRN-TAM complexity *of a shape* $\mathcal{S}$ *at temperature* τ *is the minimum complexity of any CRN-TAM program that constructs* $\mathcal{S}$*. We denote the CRN-TAM complexity of* $\mathcal{S}$ *at temperature* τ *as* $K^\tau_{\mathrm{CT}}(\mathcal{S})$*.*

Notice that since we may efficiently simulate low-temperature programs at higher temperatures, $K^\tau_{\mathrm{CT}}(\mathcal{S}) \geq K^{\tau+1}_{\mathrm{CT}}(\mathcal{S})$.

Theorem 10. *For any shape* $\mathcal{S}$*, the CRN-TAM complexity of* $s_2(\mathcal{S})$ *at any temperature* $\tau \geq 1$ *satisfies:*

$$K^\tau_{\mathrm{CT}}(s_2(\mathcal{S})) \in \Theta\left(\frac{K(\mathcal{S})}{\log K(\mathcal{S})}\right)$$

We will prove this theorem as two lemmas: Lemma 3 for the upper bound, and Lemma 4 for the lower bound.

Lemma 3. *For any shape $\mathcal{S}$, there is a CRN-TAM program $P = (S, T, R, 1, I)$ with*

$$K^1_{\mathrm{CT}}(P) \in \mathcal{O}\left(\frac{K(\mathcal{S})}{\log K(\mathcal{S})}\right)$$

that constructs $\mathcal{S}$ at scale 2.

Proof. Inspired by Soloveichik and Winfree [29], our proof gives a construction of a CRN-TAM program that satisfies the complexity bound. Our construction uses a simulated Turing machine and a finite set of "path building" tiles.

In our construction, we reduce the problem of constructing $\mathcal{S}$ at scale 2 to the problem of constructing a path around a spanning tree of $\mathcal{S}$. To do this, we use a set of tiles T_c that consists of all possible tiles with exactly two strength-1 bonds with the same label. These tiles can produce any path in the 2D lattice; the specific path is determined by the sequence in which tiles are released.

Let U be a universal Turing machine, and define ψ to be a program for U that outputs the $\mathbb{Z}^2$ coordinates of each point in $\mathcal{S}$. We construct a program φ for U that does the following:

1. Run ψ to obtain the lattice points in $\mathcal{S}$.
2. From some point in $\mathcal{S}$, use a depth-first search to find a spanning tree of $\mathcal{S}$.
3. Construct a path $W \subseteq \mathbb{Z}^2$ through the (scale 2) lattice that walks around the perimeter of the spanning tree.

Our CRN-TAM program uses the construction from Theorem 8 to simulate U running φ. We may assume without loss of generality that U acts on a binary alphabet. Then, using the implementation of U, the CRN-TAM program begins at the start of path W and releases appropriate tiles one-by-one to fill in the lattice sites occupied by W. This construction process is illustrated in Fig. 4.

Since every shape has a spanning tree, we may always find one with depth-first search. Furthermore, notice that when we scale the spanning tree to scale 2, there is always space for a perimeter walk W. Thus, P will construct $s_2(\mathcal{S})$.

Now, suppose that ψ is a Kolmogorov-optimal Turing machine program that outputs $\mathcal{S}$, so that $|\psi| = K(\mathcal{S})$. Since all parts of φ other than ψ are independent of the shape $\mathcal{S}$ and thus constant, $|\varphi| = \Theta(|\psi|) = \Theta(K(\mathcal{S}))$. By encoding φ using the optimal encoding construction in Theorem 9, which works at temperature 1, we can encode and unpack φ in $\mathcal{O}(|\varphi|/\log|\varphi|) = \mathcal{O}(K(\mathcal{S})/\log K(\mathcal{S}))$ CRN-TAM program complexity. Using the construction from Theorem 8, we may simulate the universal Turing machine U with constant program complexity. The complexity of P is the sum of the complexity of the universal Turing machine and the encoding of the optimal program, so

$$K^1_{\mathrm{CT}}(P) = \mathcal{O}(1) + \mathcal{O}\left(\frac{K(\mathcal{S})}{\log K(\mathcal{S})}\right) = \mathcal{O}\left(\frac{K(\mathcal{S})}{\log K(\mathcal{S})}\right) \qquad \square$$

Lemma 4. *For any shape $\mathcal{S}$, every CRN-TAM program $P = (S, T, R, \tau, I)$ that constructs $\mathcal{S}$ at fixed scale m has $K^\tau_{\mathrm{CT}}(P) \log K^\tau_{\mathrm{CT}}(P) \in \Omega(K(\mathcal{S}))$.*

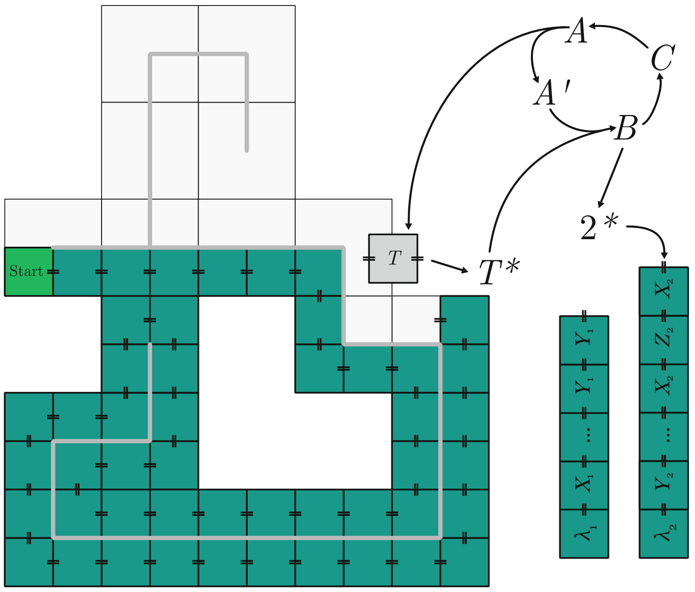

Fig. 4. Conceptual illustration of the construction of a shape $\mathcal{S}$ at scale 2 using a Kolmogorov-optimal CRN-TAM program.

Proof. First, note that there is a constant size Turing machine program p_{sim} that takes a binary description of a CRN-TAM program and simulates its operation by traversing the reachability graph of states. We construct our program so that it will output the coordinates of the occupied squares of the final assembly if the program constructs a shape. If the program stops without meeting this condition, it indicates failure.

By efficiently encoding the signals, tiles, reactions, and initial state in binary, we can represent a CRN-TAM program $P = (S, T, R, \tau, I)$ as input to our simulator in $\mathcal{O}(K^{\tau}_{\text{CT}}(P) \log K^{\tau}_{\text{CT}}(P))$ bits. By definition,

$$K(\mathcal{S}) \leq |p_{\text{sim}}| + \mathcal{O}(K^{\tau}_{\text{CT}}(P) \log K^{\tau}_{\text{CT}}(P)) = \mathcal{O}(K^{\tau}_{\text{CT}}(P) \log K^{\tau}_{\text{CT}}(P))$$

□

With Theorem 10, we demonstrate the algorithmic power of the CRN-TAM over previous models of tile-based self-assembly. Although an analogous result holds for the aTAM, it allows construction of a shape $\mathcal{S}$ only at a (possibly very) large scale c, which is polynomial in the runtime of U on ψ [29]. Constant-scale construction of algorithmic shapes with Kolmogorov-optimal tile sets is possible with temperature programming; however, these results require a number of temperature changes that is linear in the size of the shape [30]. The number of temperature changes should be a part of the natural definition of program complexity for temperature programming models. In contrast, the CRN-TAM permits construction of shapes at scale 2 with a Kolmogorov-optimal program complexity.

In the full version of this paper, we will present a result that extends this construction to give Kolmogorov-optimal assembly of a large class of shapes at scale one. It remains open to show that all shapes can (or cannot) be constructed by Kolmogorov-optimal CRN-TAM programs.

7 Open Questions

Although we have demonstrated the power and expressiveness of the CRN-TAM for Turing-universal computation and Kolmogorov-optimal construction, many open questions about the capabilities and limits of the CRN-TAM remain.

In our work, we have not considered the time complexity of computation or construction. In fact, many of our constructions proceed quite slowly under Gillespie dynamics, primarily because they take one step at a time—according to the rather limited notion of deterministic behavior used here to make our constructions simple to analyze. There are therefore numerous open questions related to the time complexity of CRN-TAM programs, such as how fast a shape can be constructed or a computation can be performed. Chemistry is an inherently parallel computational medium, yet all of our constructions have been designed to engineer around this parallelism through carefully enforced determinism. How to exploit the parallelism of chemistry to provide additional expressive power in the CRN-TAM remains to be seen.

Lastly, an important task is to develop molecular motifs that can implement CRN-TAM programs. While designs may build on the work by Zhang et al. [33], in which a DNA strand displacement circuit controlled the activation of DNA double-crossover tiles, there is a substantial difficulty with implementing the CRN-TAM based on the crystal-growth mechanism inherent in simple tile self-assembly: because (unlike in the aTAM) our model does not hold tile concentrations constant, we cannot justify low-error rates based on assuming that growth occurs near the (concentration-dependent) melting temperature for attachment by τ bonds. A more suitable molecular implementation might be based on components that become activated for further assembly by some configurational change, such as the elegant one-dimensional hybridization chain reaction [10] as generalized for a restricted class of signal tiles [15]. Such mechanisms are well suited to $\tau = 1$ seeded assembly, but have not yet been generalized for two dimensional assembly or for $\tau = 2$. We expect that the chief difficulty in a molecular implementation of CRN-TAM programs will be enforcing the proper interactions between tiles and their removal signals. However, previous work on implementing stack machines with DNA strand displacement reactions [19] proposed an implementation of a similar mechanism for handshaking assembly steps when constructing one-dimensional assemblies. Known implementations for many CRN-TAM features plausibly suggest the existence of a physical implementation of the CRN-TAM.

Acknowledgements. We acknowledge financial support from National Science Foundation grant CCF-1317694. We also thank Dave Doty, for his helpful comments and suggestions, and Kevin Li, for his useful suggestions on an early draft of this paper.

References

1. Adleman, L., Cheng, Q., Goel, A., Huang, M.D.: Running time and program size for self-assembled squares. In: ACM Symposium on Theory of Computing (STOC), pp. 740–748 (2001)
2. Aggarwal, G., Cheng, Q., Goldwasser, M.H., Kao, M.Y., de Espanes, P.M., Schweller, R.T.: Complexities for generalized models of self-assembly. SIAM J. Comput. **34**(6), 1493–1515 (2005)
3. Barish, R.D., Schulman, R., Rothemund, P.W., Winfree, E.: An information-bearing seed for nucleating algorithmic self-assembly. Proc. Natl. Acad. Sci. **106**(15), 6054–6059 (2009)
4. Bennett, C.H.: The thermodynamics of computation - a review. Int. J. Theor. Phys. **21**(12), 905–940 (1982)
5. Cardelli, L., Zavattaro, G.: On the computational power of biochemistry. In: Horimoto, K., Regensburger, G., Rosenkranz, M., Yoshida, H. (eds.) AB 2008. LNCS, vol. 5147, pp. 65–80. Springer, Heidelberg (2008)
6. Chen, H.L., Doty, D., Soloveichik, D.: Deterministic function computation with chemical reaction networks. Nat. Comput. **13**(4), 517–534 (2014)
7. Chen, Y.J., Dalchau, N., Srinivas, N., Phillips, A., Cardelli, L., Soloveichik, D., Seelig, G.: Programmable chemical controllers made from DNA. Nat. Nanotechnol. **8**(10), 755–762 (2013)
8. Condon, A., Hu, A.J., Maňuch, J., Thachuk, C.: Less haste, less waste: on recycling and its limits in strand displacement systems. Interface Focus **2**(4), 512–521 (2012)
9. Cook, M., Fu, Y., Schweller, R.: Temperature 1 self-assembly: deterministic assembly in 3D and probabilistic assembly in 2D. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 570–589. SIAM (2011)
10. Dirks, R.M., Pierce, N.A.: Triggered amplification by hybridization chain reaction. Proc. Natl. Acad. Sci. **101**(43), 15275–15278 (2004)
11. Doty, D.: Theory of algorithmic self-assembly. Commun. ACM **55**(12), 78–88 (2012)
12. Doty, D., Kari, L., Masson, B.: Negative interactions in irreversible self-assembly. Algorithmica **66**, 153–172 (2013)
13. Gillespie, D.T.: A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. J. Comput. Phys. **22**(4), 403–434 (1976)
14. Ke, Y., Ong, L.L., Shih, W.M., Yin, P.: Three-dimensional structures self-assembled from DNA bricks. Science **338**(6111), 1177–1183 (2012)
15. Padilla, J.E., Sha, R., Kristiansen, M., Chen, J., Jonoska, N., Seeman, N.C.: A signal-passing DNA-strand-exchange mechanism for active self-assembly of DNA nanostructures. Angew. Chem. Int. Ed. **54**(20), 5939–5942 (2015)
16. Patitz, M.J.: An introduction to tile-based self-assembly and a survey of recent results. Nat. Comput. **13**(2), 195–224 (2013)
17. Patitz, M.J., Schweller, R.T., Summers, S.M.: Exact shapes and turing universality at temperature 1 with a single negative glue. In: Cardelli, L., Shih, W. (eds.) DNA 17 2011. LNCS, vol. 6937, pp. 175–189. Springer, Heidelberg (2011)
18. Pinheiro, A.V., Han, D., Shih, W.M., Yan, H.: Challenges and opportunities for structural DNA nanotechnology. Nat. Nanotechnol. **6**(12), 763–772 (2011)
19. Qian, L., Soloveichik, D., Winfree, E.: Efficient turing-universal computation with DNA polymers. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16 2010. LNCS, vol. 6518, pp. 123–140. Springer, Heidelberg (2011)

20. Qian, L., Winfree, E.: Scaling up digital circuit computation with DNA strand displacement cascades. Science **332**(6034), 1196–1201 (2011)
21. Rothemund, P.W.K., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biol. **2**(12), e424 (2004)
22. Rothemund, P.W.K., Winfree, E.: The program-size complexity of self-assembled squares. In: ACM Symposium on Theory of Computing (STOC), pp. 459–468. ACM (2000)
23. Rothemund, P.W., Ekani-Nkodo, A., Papadakis, N., Kumar, A., Fygenson, D.K., Winfree, E.: Design and characterization of programmable DNA nanotubes. J. Am. Chem. Soc. **126**(50), 16344–16352 (2004)
24. Seelig, G., Soloveichik, D., Zhang, D.Y., Winfree, E.: Enzyme-free nucleic acid logic circuits. Science **314**(5805), 1585–1588 (2006)
25. Seeman, N.C.: An overview of structural DNA nanotechnology. Mol. Biotechnol. **37**(3), 246–257 (2007)
26. Sipser, M.: Introduction to the Theory of Computation. Cengage Learning, Boston (2012)
27. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. Nat. Comput. **7**(4), 615–633 (2008)
28. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. Proc. Natl. Acad. Sci. **107**(12), 5393–5398 (2010)
29. Soloveichik, D., Winfree, E.: Complexity of self-assembled shapes. SIAM J. Comput. **36**(6), 1544–1569 (2007)
30. Summers, S.M.: Reducing tile complexity for the self-assembly of scaled shapes through temperature programming. Algorithmica **63**(1–2), 117–136 (2011)
31. Wei, B., Dai, M., Yin, P.: Complex shapes self-assembled from single-stranded DNA tiles. Nature **485**(7400), 623–626 (2012)
32. Yin, P., Choi, H.M.T., Calvert, C.R., Pierce, N.A.: Programming biomolecular self-assembly pathways. Nature **451**(7176), 318–322 (2008)
33. Zhang, D.Y., Hariadi, R.F., Choi, H.M.T., Winfree, E.: Integrating DNA strand-displacement circuitry with DNA tile self-assembly. Nat. Commun. **4** (2013). Article No. 1965
34. Zhang, D.Y., Seelig, G.: Dynamic DNA nanotechnology using strand-displacement reactions. Nat. Chem. **3**(2), 103–113 (2011)
35. Zhang, D.Y., Turberfield, A.J., Yurke, B., Winfree, E.: Engineering entropy-driven reactions and networks catalyzed by DNA. Science **318**(5853), 1121–1125 (2007)