Chapter 3

A Computational Model for Active Self-Assembly in DNA Systems[0]

3.1 Abstract

In this chapter, we define the first molecularly implementable active self-assembly model. We introduce a theoretical framework for provably knowing what actions, behaviors, and life-like qualities can emerge from a given set of simple modular units. We will use some of the theoretical approaches that computer science has for determining the complexity and difficulty of solving computational problems.

There has been a need for developing formal modular theoretical models for programming active self-assembly processes in both the reconfigurable robotics community and the nanotechnology community. With respect to materials science and nanotechnology, the formal models proposed to date are either not yet implementable with our current understanding of synthetic chemistry or those that are implementable are limited to a set of features that do not capture the power of active components. Prior implementable formal models of molecular assembly only considered the passive behaviors of attaching and detaching from a molecular complex [Winfree,

[0]This work was coauthored by Nadine Dabby & Ho-Lin Chen*, and presented at the Symposium on Discrete Algorithms 2013 [Dabby and Chen, 2013b] with the following contributions: model formulation by N. D., proofs by H-L.C. ; Manuscript was written by both authors.

1996].

3.2 Introduction

In this chapter, we first discuss some of the motivation (Section 3.2.1) and contextual background (Section 3.2.2) for the work, we next review prior self-assembly models and constructions proposed across disciplines (Section 3.2.3), we then present a new "active" self-assembly model that can be directly implemented in molecules (Section 3.3) and we provide a series of theorems and proofs about what these molecules can computationally achieve (Section 3.4). The approach arises out of the fact that molecules do certain things well and other things badly, and digital computers do other types of things well and badly.

As a starting point we note that the Winfree Tile Assembly Model is a "passive" self-assembly system that formally couples computation with shape construction. It is a computational model that can be directly implemented in DNA molecules. Winfree showed that the tiles are capable of universal computation [Winfree, 1996]. Such a system is said to be "Turing-complete".

One might think that because the Tile Assembly Model is Turing-complete, capable of computing "anything," that they can do any arbitrary task. But while they can simulate any digital computational problem, there are many behaviors that are not "computations" in a classical sense, and cannot be directly implemented. Examples include exponential growth, and molecular motion relative to a surface as was discussed in Chapter 2. The tiles cannot implement these behaviors because (a) molecular motion relative to a surface requires a source of fuel that is external to the system, (b) the tiles are too slow to assemble exponentially fast growing structures and (c) the tiles are a passive self-assembling system. We call these behaviors "energetically incomplete" programmable behaviors. This class of behaviors includes any behavior where a passive physical system simply does not have enough physical energy to perform the specified tasks in the requisite amount of time. The tile computation is finished when the system passively reaches equilibrium, which is a slow process. In order to achieve these "active" behaviors, the system will need a fuel source and a logic for how it will grow or move. We will show that it does not need Turing-completeness.

As we will demonstrate and prove in this chapter, a sufficiently expressive implementation of an "active" molecular self-assembly approach can achieve these behaviors. Using an external source of fuel solves part of the the problem, so the system is not "energetically incomplete." But the programmable system needs to have sufficient expressive power to achieve the specified behaviors. Perhaps surprisingly, some of these systems do not even require Turing completeness to be sufficiently expressive.

In this chapter, motivated by some of the ideas from Chapter 1, we present a molecularly implementable idea for "active" self-assembly that exhibits behaviors such as exponential growth. To do this, we select an implementable subset of the behaviors from the Nubot model from Chapter 1 and add a programmable fuel driven approach. In this way we derive a new type of "active" self-assembly system that can be formally defined and easily implemented in molecules.

As we explained in Chapter 1, the full version of the early model cannot be implemented in molecules because it requires small groups of molecules to have more computational power than they can provide. But the right subset of behaviors (in this case exponential growth) can be programmably implemented and abstracted into a formal model, which allows one to explore the space of what can be provably constructed without entering a lab.

Although the Nubot model itself is Turing-complete, capable of performing any digital computation, the subset we have selected is not Turing-complete. Turing machines can accept or generate recursively enumerable languages. In contrast, we prove that the subset we have selected is capable of generating, at most, context-free languages (Figure 3.1). A Turing machine can rewrite its production rules, our system cannot.

Nonetheless, it is a molecularly implementable model that exhibits some of the targeted physical behaviors like exponential growth, and we prove that it exhibits these behaviors, but our system is not Turing-complete. Instead we prove that our model can generate context-free grammars and that the computational capability of this system is at most equivalent to pushdown automata.

Specifically we will show that given any insertion system (our model) we can generate both regular and context-free languages. We explore the trade-off between the complexity of the system (in terms of the number of unit types), and the behavior of the system and speed of its assembly. We find that we can grow a line of any given length n in expected time $O(log^3 n)$ using $O(log^2 n)$

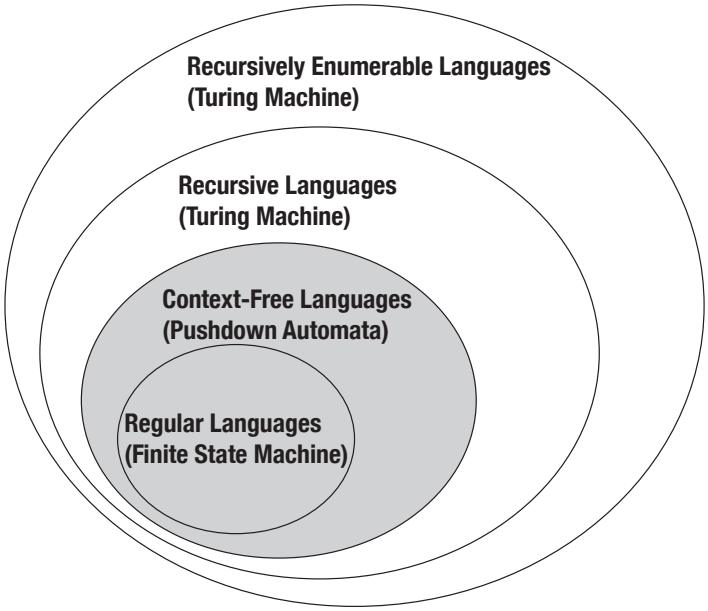

Figure 3.1: Our language is stronger than regular languages. It is within the subset of context-free languages (the shaded area in this figure) as shown in Section 3.4, but it is not as strong as recursively enumerable languages.

monomers. Finally, we show that given any insertion system with with k molecular species, either the expected final length is infinite or the expected length at time t is upper bounded by $(2t+2)^{k^2}$.

Molecular biology is missing a theoretical framework for understanding the complexity of subsets of molecules that interact with each other to generate behaviors. Computer Science has such a framework but it deals on computational complexity – thus we can say how "hard" a particular mathematical problem is by analyzing how much time and space a computer requires to solve it. On the other hand, in other parts of Biology, we can't say how computationally "hard" it is to generate behaviors like metamorphosis (the changing of one shape into another) or treadmilling (the growth of a linear polymer in one direction while it shrinks in the other direction). [1]

In the absence of biological measures of complexity we map our system onto a computational framework, by proving theorems regarding the "expressive power" of the model we define. This shows what the system is capable of doing from a computational perspective. A good primer on computer science theory can be found in [Sipser, 2006]; here we present a very simple summary.

[1] We might use the following measures to distinguish between the hardness of generating different behaviors: number of types of molecules and amount of time necessary to generate the behavior.

3.2.1 Motivation

Molecular programming, nanotechnology and synthetic biology raise the prospect of bottom-up fabrication, the manufacture of complex devices from simple components that assemble themselves. Biology sets the bar high: fabricating systems of enormous scale, defined at atomic-scale resolution, that grow quickly with small programs relative to object size and algorithmic complexity [Karsenti, 2008]. A human's genome consists of approximately three billion base pairs [Venter et al., 2001]; this implies that our cells are running a program that utilizes less than 1 gigabyte of information. Contrast this program-size efficiency with the computer on which we write this report: it has 320 gigabytes of storage disk memory, and yet it is not capable of doing many things that biology can do (e.g. it cannot grow exponentially fast like the embryos shown in Fig. 3.2, it cannot grow in mass and develop simultaneously, and it is not robust to damage). Other examples from biology prove to be even more phenomenologically interesting: a newt is able to regenerate its tail, a flatworm is capable of regenerating its head, and a starfish can regenerate its entire body from a severed leg [Alvarado and Tsonis, 2006]. Biology offers many examples of phenomena that we are as of yet unable to reproduce in computational software or hardware, but that perhaps show us what is possible. Inspired by these feats of biological efficiency, robustness and phenomena, we define a formal implementable model for active self-assembly.

Many attempts have been made to emulate biology's success across materials and disciplines. While biologists have had success reconstructing self-organized cellular systems in vitro [Liu and Fletcher, 2009], chemists have utilized self-assembly to construct films and monolayers as well as more complicated architectures constructed from nanotubes and nanoparticles [Whitesides and Grzybowski, 2002,Grzybowski et al., 2009]. These new self-assembled materials have in turn been used to construct nano-scale electronics [Lu and Lieber, 2007] and biomaterials [Stupp, 2010].

Nanotechnologists have built many examples of self-assembling two and three dimensional devices using passive subunits. The nano-components of a cell are much more "active" than passive: they sense and process environmental cues; they assemble and disassemble; upon interaction, their configurations often change, determining their future interactions; they can both diffuse and actively move. Recently, self-assembly systems using active molecular components have also

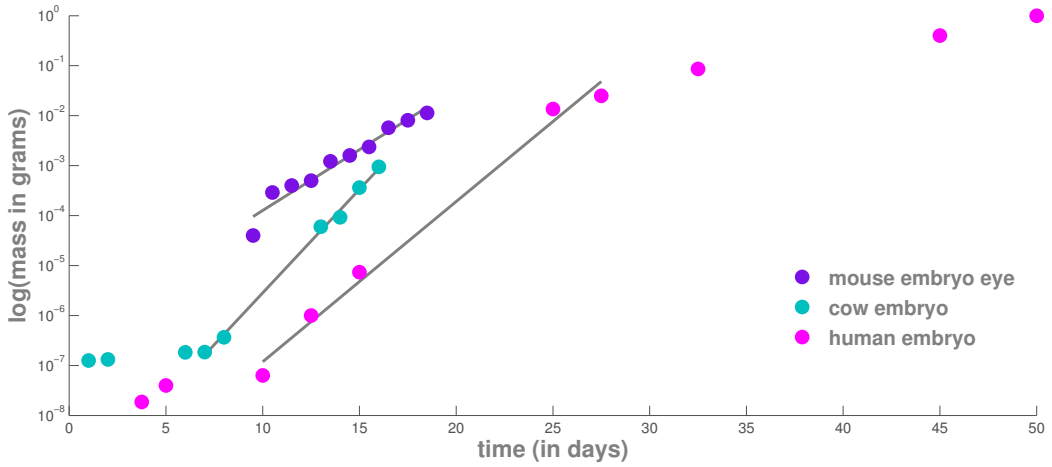

Figure 3.2: This plot is compiled from embryonic mouse [Foster et al., 2003], cow [Morris et al., 2001], and human [Luecke et al., 1999] data. The gray lines fit the periods of exponential growth in each species. Note that beginning points do not reside on these lines, because the growth rate initially increases proportionally to mass. The period of exponential growth slows down as the amount of mass necessary to sustain this type of growth becomes constrained by volume and access.

been demonstrated in various synthetic systems [Kay et al., 2007, Heuvel and Dekker, 2007, Hess, 2006a, Bath and Turberfield, 2007]. Particularly notable are the rich dynamical systems constructed out of synthetic nucleic acids, whose four-base code gives rise to a means of programming specific molecular interactions. DNA has been used to build autonomous walkers [Yin et al., 2004, Tian et al., 2005, Bath et al., 2005, Pei et al., 2006, Green et al., 2008, Omabegho et al., 2009, Yin et al., 2008, Lund et al., 2010, Muscat et al., 2011], logic and catalytic circuits [Seelig et al., 2006, Zhang et al., 2007, Yin et al., 2008, Win and Smolke, 2008], and triggered assembly of linear [Dirks and Pierce, 2004, Venkataraman et al., 2007] and dendritic structures [Yin et al., 2008].

Now that our once passive subunits can actively sense, walk and otherwise actively interact, how do these new "rules" change the prospects for what we can build from the bottom-up? This notion of actively assembling molecules is already an experimental reality, but as of yet there is no satisfying theory to guide future work. In this paper we attempt to formulate a framework for integrating these new "active" mechanisms in nanotechnology to build "programs" that can grow into a desired shape quickly and with relatively small program size to a final structure.

3.2.2 Contextual Background

In computer science, one can divide up the difficulty of a computational problem by classifying the strength of the machine necessary to solve it; these machines are associated with the types of languages they are capable of accepting as input or generating. A language is a set of strings of symbols that can be generated by a set of production rules [Sipser, 2006].

The simplest classes of machines are finite automata. For example one can think of the motion sensing doors in the supermarket– the doors are capable of being in two states, open or closed, and the doors go between these states with the various inputs to the motion sensors. The system has an extremely limited memory. We say a machine "accepts" a language if the machine can take a string from that language as input and end in one of its terminal states. A finite automata is capable of accepting regular languages as input.

The next more complex class of machines are called pushdown automata; these are essentially finite state machines that have an additional stack memory. Pushdown automata are capable of accepting context-free grammars as input to the machine.

Both finite state machines and pushdown automata are less complex than Turing machines. Turing machines have multiple states, memory and the ability to conduct if-then logic. With these key capabilities, Turing machines can accept all unrestricted grammars (grammars that have no restrictions on either the right or left side of the grammar's production rules) and they can generate arbitrary recursively enumerable languages and are capable of solving all computable problems. Our system is not as strong as a Turing machine, rather it can generate languages that are at most equivalent to context-free grammars.

Our theorems and proofs below are derived from our ability to formalize our model into a language that is, at most, as strong as a context-free grammar. A context-free grammar is a class of formal languages that can be generated by production rules. Here we use the formal definition of a context-free grammar taken from [Sipser, 2006]:

A context-free grammar is a 4-tuple (V, Σ, R, S) such that

1. V is a finite set of variables.

2. Σ is a finite set, disjoint from V, of terminals.

3. R is a finite set of rules, where each rule takes a variable and transforms it into a string of variables and terminals.

4. $S \in V$ is the start variable.

3.2.3 Review of Self-Assembly Models

The Tile Assembly Model integrates the algorithmic association of units with a defined geometry: the exposed edges of a growing crystal encode the state information of the system, and this information is modified as a new tile attaches itself to the crystal [Winfree, 1996]. This model formally couples computation with shape construction, and the shape can be viewed as the output of the tile assembly "program". Tiles are capable of universal computation [Winfree, 1996]. The system can grow an arbitrary shape (independent of scale) using a tile program whose complexity, defined as the number of distinct tile species in the program, is bounded from both above and below by the shape's descriptional (Kolmogorov) complexity [Soloveichik and Winfree, 2005]. The time required to build an $n \times n$ shape through passive self-assembly is $O(n)$ [Adleman et al., 2001]. This bound can be improved to $O(n^{4/5}log(n))$ with massive parallelism [Chen and Doty, 2012]. In this model, scale plays the same role in the self-assembly process as time plays in computability. While the Tile Assembly Model is elegant in its simplicity and ability to capture experimental reality, it is limited in its speed, its ability to be scaled up and its focus on passively assembling units.

Drawing on cellular automata and Chomsky grammars, L-systems were developed as a theoretical framework for studying development in multicellular organisms and were one of the first models used to simulate growth and development in plants [Lindenmayer, 1987]. Although they bear a resemblance to cellular automata, they differ in that arrays can grow and shrink (introducing the notions of insertion, a new cell is generated by division of a prior cell, and deletion, the elimination of a cell). L-systems differ from grammars in that they require parallel rewriting of all symbols and do not distinguish between terminal and non-terminal symbols [Lindenmayer, 1987]. While these models are well-developed for one-dimensional systems, they have also been studied in two [Siromoney, 1986] and three dimensions [Prusinkiewicz, 2004]. While L-systems have aided in the modeling of plant growth and biology, the formal work does not address theoretical

questions related to the complexity of pattern formation such as how quickly a system can generate a specific pattern.

A number of geometric models and numerous algorithms have been described for self-assembling and reconfigurable modular robotic systems [Chirikjian, 1993, Goldstein and Mowry, 2004, Rosa et al., 2006, Griffith, 2004, White et al., 2004, Jones and Mataric, 2003, Murata et al., 1994, Nagpal, 2008, Werfel and Nagpal, 2007, Arbuckle and Requicha, 2004, Rus and Vona, 2001, Butler et al., 2001, Yim et al., 1997, Yim et al., 2007, Groß and Dorigo, 2008, Walter et al., 2004]. Existing formal models have not fully captured the efficiency of active self-assembly: to assemble a prescribed shape, most of the models require a linear (to the size of the shape) number of distinct states.

One of the central questions that this work addresses is how to program global tasks through local interactions. Our approach is inspired by Klavins' work on modeling robotic self-assembly [Klavins et al., 2004] using conformational switching [Saitou, 1999] and graph grammars [Ehrig, 1979]. In Klavins' work, the state of a physical system is represented as an abstract graph, where an assembly unit is represented as a symbolic vertex labeled with its current state, and the attachment of two units is represented by an edge connecting the two corresponding vertices in the graph. Assembly proceeds following graph rewriting rules, which update the system state by updating the vertex labels and edges of a subgraph under suitable conditions. This approach nicely captures the local, asynchronous, cooperative and conditional state change logic, which is intrinsic to assembly systems with active components, and it captures disassembly in addition to assembly. However, unlike the Tile Assembly Model, the graph grammar model represents the assembly system as an abstract graph, and leaves out geometry, which is a crucial property for the assembly of physical systems.

In our prior work on active self-assembly, we constructed the "nubot" model by adding a geometric component to the graph grammar model [Woods et al., 2013]. The nubot model builds on the concept of graph grammars, by defining rule sets over two dimensional monomers, represented as disks of unit diameter centered on a point in a hexagonal grid. Two monomers can react with each other (according to a rule) to change state, make and break bonds, change relative position, appear and disappear. With this model, a line of length n can be constructed with $O(\log n)$ states, in $O(\log n)$ time. A computable shape of size $n \times n$ pixels can be constructed in time polyloga-

rithmic in n. This is exponentially faster than systems composed entirely of passive components. While the nubot model is not chemically implementable, it highlights the fundamental efficiency advantage of active self-assembly over passive self-assembly. We seek to preserve the complex behaviors that the abstract nubot model can generate, but in a formulation that is simple enough to implement experimentally.

3.3 Model

3.3.1 Formal Model Description

In our model, each construction begins with an initiator, and grows via the insertion of simple units that we call monomers. We assume that each type of monomer in the system is present in infinite amounts. Monomers can be inserted into the middle of the structure and increase the length of the structure (an abstraction of the model is shown in Fig. 3.3). Figure 3.4 shows an example system that grows exponentially fast. The detailed description of initiators, monomers, and the insertion rules follows:

1. We have two finite sets of symbols $\Gamma = \{a_1, a_2, a_3, a_4, \dots\}$ and $\Gamma^* = \{a_1^*, a_2^*, a_3^*, a_4^*, \dots\}$. Each pair a_i and a_i^* are called *complementary* to each other.

2. There are k monomers, each is described by a quadruple of symbols (a, b, c, d) and either a plus sign or a minus sign. The plus and minus sign indicate the directionality of the molecules and are used in mapping the model onto a direct DNA implementation, which requires both $5'$ and $3'$ sequences. (For example, $(a_4, a_7, a_6^*, a_1)+$ or $(a_5, a_7, a_2^*, a_3^*)-$.) Each monomer has a concentration c. We assume that the total concentration is at most 1.

3. The initial state can be described by two pairs of symbols $(a, b), (c, d)$. Either a and d are complementary to each other or b and c are complementary to each other. Each of these pairs is considered a monomer.

4. An *insertion site* can only exist between two consecutive monomers: e.g., in the initial state (a, b) and (c, d) belong to two different monomers.

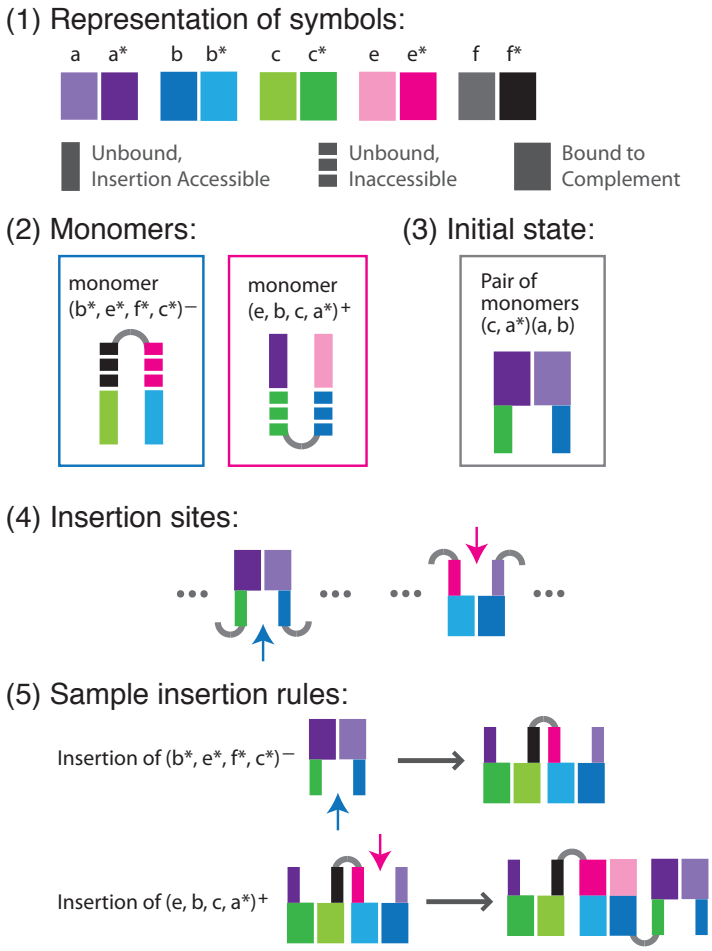

Figure 3.3: This figure depicts an abstraction of our model. (1) Each unique symbol is encoded by a color, and complementary symbols are represented by different shades of the same color. The symbols are represented as thin solid lines (Unbound, Insertion Accessible), thin dashed lines (Unbound, Inaccessible), and thick solid lines (Bound to Complement). (2) Two sample monomers are $(b^*, e^*, f^*, c^*)-$, and $(e, b, c, a^*)+$. (3) The initial state is described by the pair of doubles $(c, a^*), (a, b)$. (4) Insertion sites can only exist between two consecutive monomers connected in the structure; we use colored arrows to denote these sites. (5) Sample insertion rules show the insertion of monomer $(b^*, e^*, f^*, c^*)-$ into $(c, a^*), (a, b)$ to generate the polymer $(c, a^*), (f^*, c^*, b^*, e^*), (a, b)$, and the insertion of monomer $(e, b, c, a^*)+$ into $(c, a^*), (f^*, c^*, b^*, e^*), (a, b)$ to generate the polymer $(c, a^*), (f^*, c^*, b^*, e^*), (e, b, c, a^*), (a, b)$.

5. Only the following insertion rules are possible:

   (a) If there are two consecutive monomers connected in the structure such that the first one ends with the pair (e, a^*) and the second one starts with the pair (d^*, f), where e and f are complementary with each other, then any monomer of the form $(a, b, c, d)+$ can insert between those two groups, and add a group of symbols (a, b, c, d) in the middle. $(e, a^*), (d^*, f)$ is called an *insertion site*.

   (b) If there are two consecutive monomers connected in the structure such that such that the first one ends with (d^*, e) and the second one starts with (f, a^*), where e and f are complementary with each other, then any monomer of the form $(a, b, c, d)-$ can insert between these two groups and add a group of symbols (c, d, a, b) in the middle. $(d^*, e), (f, a^*)$ is called an *insertion site*.

6. If a particular insertion is applicable, it occurs at time x, where x is an exponential random variable with rate c, where c is the concentration of the monomer inserted.

7. A *polymer* is a sequence of tuples of symbols reachable from the initial state, where the first and last tuples are pairs of symbols and the middle tuples are monomers (as defined in rule 2). A *terminal polymer* is a polymer such that no monomers exist in the system that can be inserted at any of the insertion sites available on that polymer. The *length* of the polymer is defined as the number of monomers that it contains.

3.3.2 A Molecular Implementation

Given any system described above, there is a direct implementation of monomers into a set of DNA molecules. By encoding the order of the nucleotides in a DNA sequence, we can control the interaction of DNA strands. Subsequences of these strands are called domains and it is their binding (hybridization) and unbinding (disassociation) from complementary domains that determines what a system can do. In DNA nanotechnology, dynamic systems of DNA molecules can be controlled by toeholds, the short sequences of DNA that are complementary to single stranded

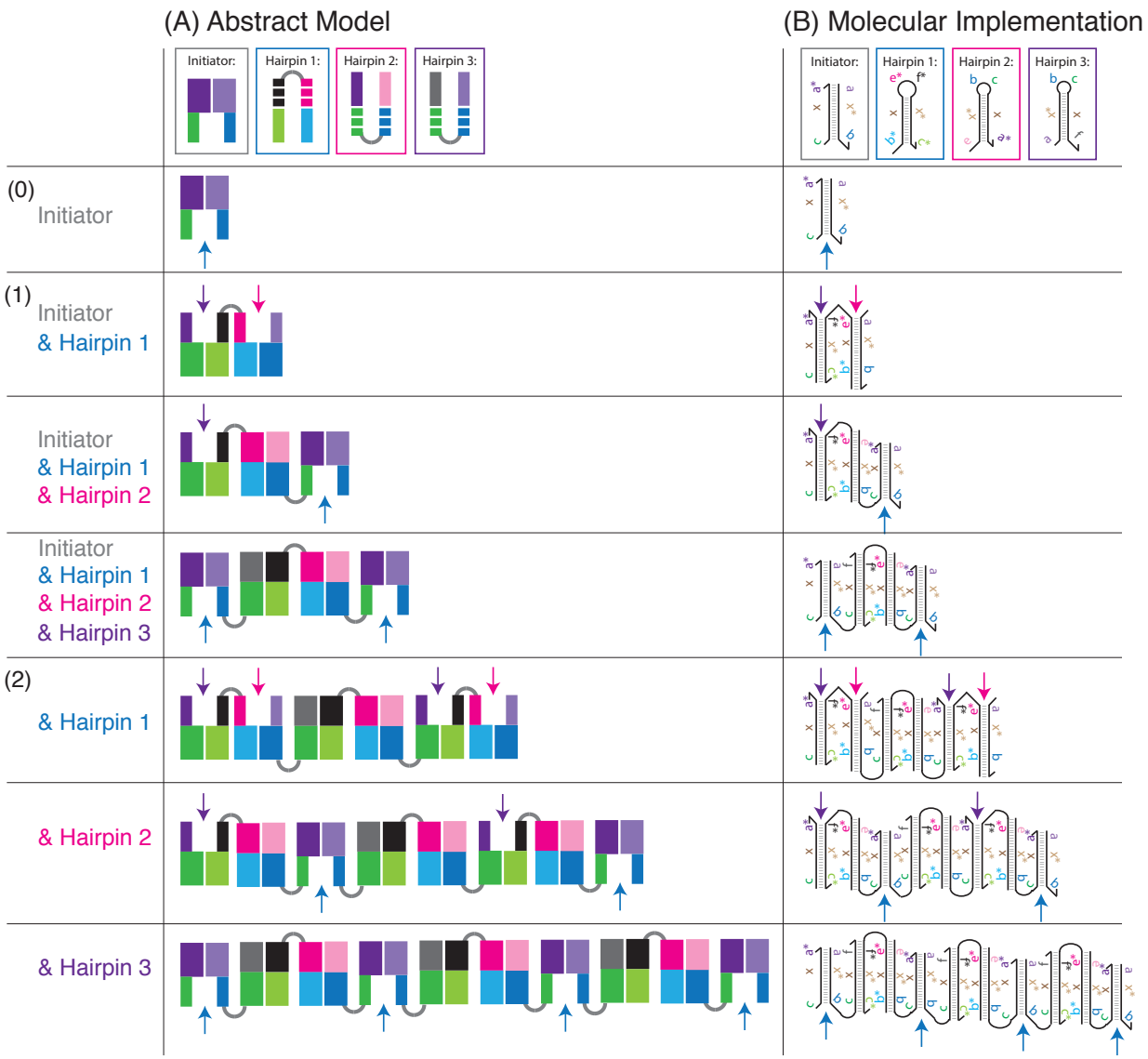

Figure 3.4: This figure depicts a system that implements insertional polymer growth in logarithmic time. The abstract representation of growth (A), is directly correlated to a molecular implementation (B). In this insertion system, the initiator is described as $(c, a^*), (a, b)$ and the three hairpins are $(b^*, e^*, f^*, c^*)-$, $(e, b, c, a^*)+$, and $(a, b, c, f)+$ After inserting hairpin 1, the polymer's description is $(c, a^*), (f^*, c^*, b^*, e^*), (a, b)$. After hairpins 2 and 3 are inserted, the polymer's description is $(c, a^*), (a, b, c, f), (f^*, c^*, b^*, e^*), (e, b, c, a^*), (a, b)$. The system will continue to grow to infinite length exponentially fast.

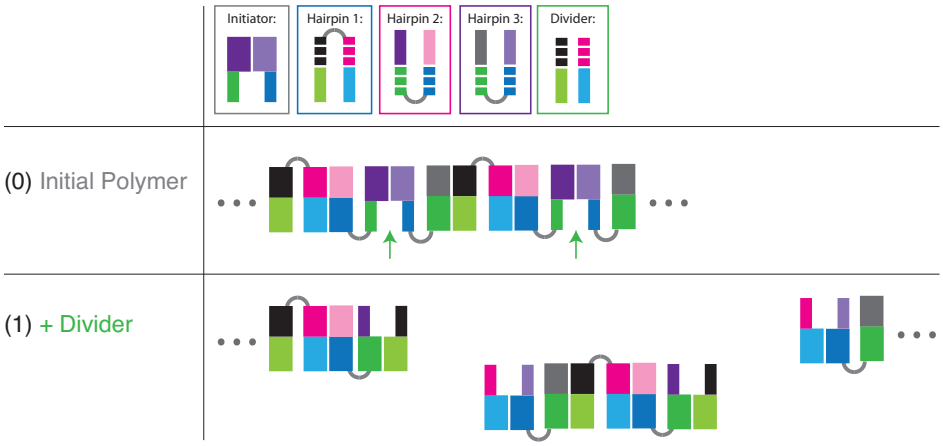

Figure 3.5: This figure depicts a system that implements division in a polymer. The reaction available for $(a, b)(c, d)+$ is exactly the same as that for $(a, b, c, d)+$, except that after $(a, b)(c, d)+$ inserts, the polymer will be cut between (a, b) and (c, d) and divided into two parts.

domains in a target molecule [Yurke et al., 2000, Zhang and Winfree, 2009]. Toeholds serve as the inputs to dynamic DNA systems and initiate branch migration processes, the random walk process of bond breaking and formation that results in the exchange of one strand in the duplex for another single strand with the same sequence. Our DNA implementation (Figure 3.4) is inspired by the Hybridization Chain Reaction system developed by Dirks and Pierce [Dirks and Pierce, 2004] and will be discussed in depth in Chapter 4.

Any system described in our model can be implemented by designing DNA hairpins and an initiator complex as follows:

For every monomer $(a, b, c, d)-$, we add a hairpin with domains (a, x, b, c, x^*, d), where x (composed of 18 bases) is the long stem of the hairpin. For every monomer $(a, b, c, d)+$, we add a hairpin with domains (a, x^*, b, c, x, d). The initiator is (a, x^*, b) binding with (c, x, d). The insertion rules defined in the model correspond to all possible reactions that can happen in the corresponding molecular system. In addition to the monomer $(a, b, c, d)+$ (or minus), we can also have a new type of monomer $(a, b)(c, d)+$ that we call a divider monomer. The reaction available for $(a, b)(c, d)+$ is exactly the same as that for $(a, b, c, d)+$, except that after $(a, b)(c, d)+$ inserts, the polymer will be cut between (a, b) and (c, d) and divided into two parts , as will be described in Chapter 4.

3.4 Proofs of the Model's Expressive Power

In this section, we first ignore the rates of insertion and show that the expressive power of this insertion system is, at most, equivalent to context-free languages. This result implies that we can simulate arbitrary tile systems that assemble a single line. From [Becker et al., 2006], we know that the insertion system can construct lines of arbitrary expected length with $O(1)$ monomers.

Theorem 1 *Given any insertion system, the set of terminal polymers that can be generated forms a context-free language.*

Proof: Given any insertion system with n symbols, the corresponding context-free language has n^4 symbols, each of which corresponds to one insertion site. The starting symbol S corresponds to $(a, b), (c, d)$, which is the initiator of the polymer. Each monomer $(e, f, g, h)+$ corresponds to $2n$ different production rules in the context-free language that starts with a symbol (insertion site) $(i, e^*), (h^*, j)$ and produces two symbols $(i, e^*), (e, f)$ and $(g, h), (h^*, j)$ for all possible choices of pairs of complementary symbols i, j in the insertion system. □

Theorem 2 *Given any regular language, there is an insertion system that generates terminal polymers corresponding to this language.*

Proof: Given any left regular grammar with nonterminal symbols $A_1, A_2, \ldots, A_n$, including the starting symbol A_1, and non-terminal symbols $\alpha_1, \alpha_2, \ldots, \alpha_m$, the following insertion system creates polymers that correspond to the given regular language:

1. $\Gamma = \{a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n, c_1, c_2, \ldots, c_m, d\}$.
2. The initiator is $(d^*, a_1), (d, d)$.
3. For each production rule $A_i \rightarrow \alpha_j A_k$, there are two corresponding monomers $(a_i^*, c_j, b_k, d^*)+$, $(d^*, a_k, d, b_k^*)-$.

In this system, there is always exactly one insertion site available at the end of the polymer. The insertion can only happen between two monomers (d, b_i^*, d^*, a_i) and (d, d). The insertion

site between these two monomers corresponds to the nonterminal symbol A_i. At this point, two monomers $(a_i^*, c_j, b_k, d^*)+$, $(d^*, a_k, d, b_k^*)-$ may insert, generate some inactive sequence with j encoded in the middle, and the end of the polymer becomes (d, b_i^*, d^*, a_k) and (d, d), corresponding to the nonterminal symbol A_k. □

Corollary 1 *There is a family of insertion systems that can construct polymers of expected length n with $O(1)$ monomers.*

Proof: Since insertion systems are able to simulate all regular languages, they are able to simulate all tile systems that form a linear polymer of width 1. Therefore, the proof directly follows from [Becker et al., 2006], where the result was proven on 1-dimensional tile assembly systems. □

3.4.1 Analyzing the Theoretical Growth Speed of Polymers

We also investigate the speed at which these polymers can be constructed. First, we show that arbitrarily long polymers can be constructed deterministically in expected polylogarithmic time using a polylogarithmic number of monomers.

Lemma 3 *The following insertion system deterministically constructs a line of length $n = 2^{2k} + 1$ in expected time $O(\log^3 n)$ and only uses $O(\log^2 n)$ monomers. Furthermore, the required time has an exponentially decaying tail probability.*

1. *The initiator is $(c, a_{2k}), (b_{2k}^*, c^*)$.*
2. *For every $i, j \in \{2, 4, \ldots, 2k\}, i \leq j$, there are two monomers $(a_i^*, b_{i-1}^*, a_{i-1}, b_j)+$ and $(a_j^*, b_{i-1}^*, a_{i-1}, b_i)+$. For every $i \in \{1, 3, \ldots, 2k-1\}$, there are two monomers $(b_i, a_{i-1}, b_{i-1}^*, c^*)-$ and $(c, a_{i-1}, b_{i-1}^*, a_i^*)-$.*
3. *All monomers have equal concentration $\frac{1}{2k^2}$.*

Proof: First, we show that the system deterministically constructs a line of length n. An insertion site of the form $(c, a_i), (b_j^*, c^*)$ is defined to have type $\min\{i, j\}$. Whenever a gap of type i is available, exactly one monomer of the form $(a_i^*, b_{i-1}^*, a_{i-1}, b_j)+$ and $(a_j^*, b_{i-1}^*, a_{i-1}, b_i)+$

will be able to attach. After the first monomer inserts, two monomers $(b_{i-1}, a_{i-2}, b^*_{i-2}, c^*)-$ and $(c, a_{i-2}, b^*_{i-2}, a^*_{i-1})-$ will be able to insert on the first monomer's left and right. These three insertions create four insertion sites of type $i-2$. Therefore, starting with one insertion site of type k on the initiator, there will be $2^k - 1$ total insertions, resulting in a polymer that has n insertion sites of type 0. At that time, no further insertion is available and the system halts.

Second, the system halts as soon as all $\frac{n}{2}$ insertions of $(b_1, a_0, b^*_0, c^*)-$ and $(c, a_0, b^*_0, a^*_1)-$ happen. Each of these insertions only relies on k insertions to occur before them. Therefore, for any one of these insertions, the expected time T until the insertion occurs can be described as a sum of k independent exponential random variables of expected values $2k^2$. Using Chernoff bounds for exponential random variables, it follows that

$$\text{Prob}[T > 2k^2 \cdot k(1+\delta)] \;\leq\; (\frac{1+\delta}{e^\delta})^k.$$

Although the times required for these $\frac{n}{2}$ insertions are not independent of each other, we can still use a union bound to get the following bound for the total running time T_{fin} of the system:

$$\begin{aligned}\text{Prob}[T_{fin} > 2k^2 \cdot k(1+\delta)] \;&\leq\; \frac{n}{2}(\frac{1+\delta}{e^\delta})^k \\ &\leq\; \frac{n}{2}e^{-\frac{k\delta}{2}} \;<\; (\frac{e}{2})^{-\frac{k\delta}{2}}, \text{ for all } \delta > 4.\end{aligned}$$

Therefore, the expected time is $O(k^3) = O(\log^3 n)$ with a tail probability that exponentially decays. □

Theorem 4 *There exists an insertion system that deterministically constructs a line of length n in expected time $O(\log^3 n)$ and only uses $O(\log^2 n)$ monomers for every integer n. Furthermore, the required time has an exponentially decaying tail probability.*

Proof: Lemma 3 already showed that the theorem is true for all $n = 2^{2k} + 1$. Given an arbitrary n, we can write $n - 1$ as the sum of $O(\log n)$ terms $2^{r_1} + 2^{r_2} + \cdots + 2^{r_m}$., where all r_is are even numbers. We can first construct m distinct monomers that must insert one by one at the beginning, creating m insertion sites identical to the initiator for a polymer of length $2^{r_i} + 1$. Afterwards,

the system described in Lemma 3 can make a line of length n. Since m is only $O(\log n)$, this construction works in the required $O(\log^3 n)$ time and $O(\log^2 n)$ monomers. □

In the rest of this section, the major goal is to show that for an insertion system with k different molecular species, either the expected final length is infinite, or the expected length grows polynomially with time.

Theorem 5 *Consider a context-free language L with m symbols (including terminal and nonterminal symbols) in reduced Chomsky normal form. When a production rule is applicable, the time until it is applied is a random variable of rate k. If, for any given symbol A, the rate of all production rules having A on the left side sum up to at most 1, then either the expected final length is infinite, or the expected number of symbols at time t is upper bounded by $(2t+2)^m$.*

Proof:

Assuming the expected final length is finite, we will prove inductively on m that the expected number of symbols at time t is upper bounded by $(2t+2)^m$

The general idea is that starting with S, we can't produce S too fast, otherwise the expected length will become infinite. Furthermore, since L is a context-free language, if all we want to know is the length of the string, we only need to keep track of how many copies of each symbol is currently in the string. If each time we generate S we isolate that symbol into a new sub-system, then each sub-system is essentially a system with $m-1$ different symbols and the growth speed will be bounded by the induction hypothesis.

The theorem is true when $m = 2$. Since there are only two symbols S and α, starting from S, if the rate of the production rule $S \to SS$ is higher than the rate at which $S \to \alpha$, then the expected length is infinite. Otherwise the expected length is linear in t, since the expected number of symbols S in a string is at most 1 at any given time.

Assume that the theorem is true for $m = k-1$. For $m = k$, we subdivide the sets of symbols into many subsets in the following way: initially, there is only one subset that contains S; whenever one copy of S gets produced in any subset, we move that symbol S into a new subset; when other types of symbols are produced, they stay in the same subset.

First, we show that the expected number of symbols in each subset is quite small at time t. We start by considering the subset T_1 that the initial symbol S belongs to. After applying the first production rule, the subset T_1 has at most 2 symbols and will never contain another copy of S again. Therefore, after that first production rule, only $k-1$ different symbols can appear in that subset. By the induction hypothesis, either the expected number of symbols in T_1 goes to infinity, or the expected number of symbols is upper bounded by $2 \cdot (2t+2)^{k-1}$. The exact same argument can be applied to all other subsets.

Second, we will show that the expected number of subsets at time t is at most t. Notice that the number of subsets is equal to the number of symbols S that have been generated in the process. For the expected final length to be finite, the expected number of symbols S in the system is at most 1 at any given time. (Otherwise the number of symbols S is expected to grow exponentially, a contradiction.) Furthermore, since the total rate of all rules with S on the left side is 1, the expected rate at which S is removed by applying production rules is also at most 1 at any time. Therefore, at any time t, the expected number of symbols S that have been removed by a production rule is at most t. Combining the above arguments, the expected number of symbols S that have ever appeared in the system before time t is at most $t+1$.

According to our definition, at time t, the expected number of subsets is equal to the expected total number of symbols S that have ever appeared in the system, which is at most $t+1$. Also, the expected number of symbols in each subset is at most $2 \cdot (2t+2)^{k-1}$. Using linearity of expectation, we know that the expected number of total symbols at time t is at most $(2t+2)^k$.

□

Corollary 2 *Given any insertion system with k molecular species and total concentration 1, either the expected final length is infinite, or the expected length at time t is upper bounded by $(2t+2)^{k^2}$.*

Proof: There are at most k^2 different insertion sites in a system with k species. From Theorem 1, we know that the insertion system can be described by a context-free grammar in reduced Chomsky normal form with at most k^2 symbols. Therefore, the proof follows from Theorem 5. □

3.5 Conclusions

We have defined a formal implementable model for active self-assembly utilizing an insertion primitive. We build on the concept of applying biological algorithms to the development of novel techniques in computer science to provide a method by which we can program arbitrary insertion systems whether they be reconfigurable robots, molecules or scripts of symbols. The work here is particularly relevant for the application of computer science to synthetic biology, chemistry and material science. We show a construction for building a line in polylogarithmic time using a polylogarithmic number of monomers and map it to a molecular system. To our knowledge this is a novel assembly system that has never been synthetically constructed before. We also show that with a number of monomer types the system will either grow to infinity or the expected length of the polymer grows polynomially with time. There are many interesting open questions remaining: What other behaviors can be generated by such a simple model? Are there other directly implementable simple primitives that we can add to this model to generate such behaviors? In this chapter we explored the expressive power of this language, and proved that the language is stronger than regular languages, but, at most, as strong as context-free grammars. It remains to be shown whether this system is equivalent to context-free grammars, in which case the language will prove to be even more powerful than we suggest here.