5-2022

Verification in Generalizations of the 2-Handed Assembly Model

David Caballero
The University of Texas Rio Grande Valley

VERIFICATION IN GENERALIZATIONS OF THE 2-HANDED ASSEMBLY MODEL

A Thesis

by

DAVID CABALLERO

Submitted in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: Computer Science

The University of Texas Rio Grande Valley

May 2022

VERIFICATION IN GENERALIZATIONS OF THE 2-HANDED ASSEMBLY MODEL

A Thesis
by
DAVID CABALLERO

COMMITTEE MEMBERS

Dr. Tim Wylie
Chair of Committee

Dr. Robert Schweller
Committee Member

Dr. Zhixiang Chen
Committee Member

Dr. Bin Fu
Committee Member

May 2022

ABSTRACT

Caballero, David E., Verification in Generalizations of the 2-Handed Assembly Model. Master of Science (MS), May, 2022, 75 pp., 1 table, 30 figures, 36 references.

Algorithmic Self Assembly is a well studied field in theoretical computer science motivated by the analogous real world phenomenon of DNA self assembly, as well as the emergence of nanoscale technology. Abstract mathematical models of self assembly such as the Two Handed Assembly model (2HAM) allow us to formally study the computational capabilities of self assembly. The 2HAM is one of the most thoroughly studied models of self assembly, and thus in this paper we study generalizations of this model. The Staged Tile Assembly model captures the capability of being able to separate assembly processes and combine their outputs at a later time. The k-Handed Assembly Model relaxes the restriction of the 2HAM that only two assemblies can combine in one assembly step. The 2HAM with prebuilt assemblies considers the idea that you can start your assembly process with some prebuilt structures. These generalizations relax some rules of the 2HAM in ways which reflect real world self assembly mechanics and capabilities. We investigate the complexity of verification problems in these new models, such as the problem of verifying whether a system produces a specified assembly (Producibility), and verifying whether a system uniquely assembles a specified assembly (Unique Assembly Verification). We show that these generalizations introduce a high amount of intractability to these verification problems.

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisors Dr. Robert Schweller and Dr. Tim Wylie. If not for them I likely never would have been introduced to the world of research. They have been kind and supportive as advisors as well as professors, and have created a friendly and welcoming research lab that allows student to reach their academic potential. I would like to thank Timothy Gomez, a friend and colleague who has been there throughout my academic research career. I appreciate his friendship, support, and for inspiring other students and I to be a better researcher. Additionally I thank all the members of the Algorithmic Self Assembly Research group. The group is filled with amazing students who never fail to offer kindness and support. I also thank them for all their hard work work that has allowed us to achieve the results we have.

I thank Dr. Bin Fu and Dr. Zhixiang Chen for serving on my thesis committee, as well as for the teaching they provided throughout my studies. Bin Fu provided great instruction throughout many of my graduate classes, and Zhixiang Chen provided me with challenging classes in my undergraduate studies that brought out the best of my abilities.

I would like to thank my mother Erica Caballero for her constant love and support. She has always been there when I needed her and I couldn't ask for a better mother. Finally I would like to thank all my friends and family for all their support and the companionship that kept me sane along the way.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER I

INTRODUCTION

Self assembly is the process in which simple individual units come together to autonomously build complex structures utilizing only local interactions. We can see this process happen at the nanoscale level in processes such as DNA self assembly. It has been shown how the power of DNA self assembly can be harnessed and used for the construction of nanoscale structures such as 2D and 3D lattices [30][36] as well as polyhedral framework structures [20]. The creation of DNA Double-Crossover Molecules, a DNA tile implemented with DNA double helices, allowed for a natural abstraction to tile self assembly[19].

We have also seen the emergence of self assembly in the field of robot swarms. Having a large number of robots is feasible only when each individual robot is of limited complexity. The limited capabilities of these robots have progressed the use of self assembly techniques in their interactions. This allows the individual robots to remain as very simple components with limited computational power, while allowing the swarm to have a high complex capabilities. Work has shown that these self assembly techniques allow for complex coordination and communication in robot swarms [27][24].

The field of algorithmic self assembly is the study of abstract mathematical models of self assembly. Research in the field primarily focuses on modeling and studying the computation power of self assembly processes. This was first introduced in the PhD thesis of Erick Winfree in 1998 [31]. The Abstract Tile Assembly Model (aTAM) is the first of these to be formally named, defined and studied [1]. In this model unit squares can attach to each other based on

defined affinity labels on their four cardinal directions. These squares build on to a *seed* structure if their affinity labels allow it, allowing for the creation of complex structures. The Two Handed Assembly Model (2HAM), alternatively known as the Hierarchical Self Assembly Model is different in that it defines how larger assemblies can attach to each other, as opposed to just single tiles [10]. The 2HAM captures a natural capability that the aTAM does not, that naturally two complex assemblies could attach to each other as opposed to just single tiles attaching to a seed assembly. It has been discovered that these self assembly systems have a wealth of computational power, despite their seemingly simple nature. For example, both of the aforementioned models are Turing universal.

With a formal and rigorously defined model, the capabilities of these self assembly models can be formally studied. A common area of study in self assembly is that of shape building. This is the study of how efficiently, in terms of number of tiles necessary, a shape can be uniquely self assembled. This has led to the development of innovative techniques utilizing the computational capabilities of self assembly systems. Work has shown that by implementing binary counters and Turing machine simulations, shapes such as squares can be self assembled with extreme efficiency [1][2].

Another area of study in self assembly models is the study of verification problems. Verification problems focus on verifying the behavior of a self assembly system. After the design of a self assembly system intended to build some structure, it is desirable to have an algorithm that verifies it does what is intended. It would be nice to know that the system you just designed does in fact uniquely assemble a square assembly, and not something else in addition that was not intended. This has shown that verifying the behavior of a self assembly can be a drastically different challenge with only just a simple change to a model. For example verifying the behavior of an aTAM system can be solvable in polynomial time, or undecidable, depending on if negative strength glues are allowed [6][7].

The aTAM and 2HAM have been thoroughly studied, with few open problems remaining related to verifying system behavior. In this work we investigate models that generalize on the 2-Handed assembly models in ways that reflect real world capabilities and behavior in self assembly systems. We will now at a high level discuss the 2HAM and the generalized models of focus.

1.1 The Two Handed Assembly Model

The two handed assembly model is a well studied model of self assembly. In this model the smallest unit is a a four sided tile with glue labels on every edge. A 2HAM system is a defined set of tiles, the glue function that determines the strength of each glue, and a threshold, or temperature which specifies the strength required for two assemblies to attach. An assembly step in the 2HAM is when any two assemblies in the system meet, if their edges can align without overlapping such that the strength of their matching adjacent glues surpasses the threshold, then the union of these two assemblies now exists in the system (Figure 1.3).

1.2 k-Handed Assembly Model

One generalization of the 2HAM which has not been thoroughly explored is the k-Handed Assembly model. One feature of the 2HAM is that two and *only* two assemblies can come together in one assembly step. The k-handed assembly model provides the power of defining how many assemblies can come together in one assembly step, also called the "handedness" of the model. This reflects a natural mechanic of self assembly that, though less probable, more than 2 assemblies could align in such a way that they combine to form a structure that theoretically would not be produced when abiding by the rules of the 2HAM. This allows us to study the consider how a system might act in practice, with the handedness of the model denoting the level of improbability we wish to consider. See Figure 1.2 for an example.

1.3 The Staged Tile Assembly Model

The Staged Tile Assembly model is another generalization of the 2HAM. Staged Tile Assembly uses the same assembly logic as the 2HAM, where any two assemblies can attach if their adjacent glues meet the threshold. There exists an initial set of bins each of which has a defined 2HAM tile set. Assembly occurs in each bin as it would in 2HAM, and the terminal assemblies of a bin are passed into new bins in a second according to a mixgraph. The producibles in bins after stage 1 consist of the terminal assemblies passed into that bin, and the process continues (See Figure 1.1). The Staged Tile Assembly Model reflects the natural capability of separating assembly processes.

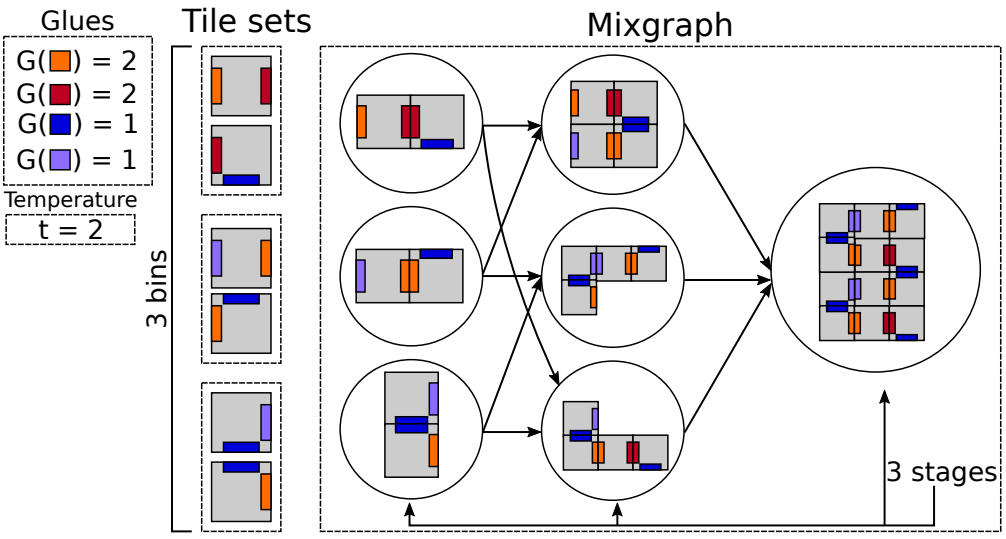


Figure 1.1: Example Staged Tile Assembly system. This system has 3 stages and 3 initial bins, with 2 tile types in each bin. The second stage has 3 bins, the arrows in the mix graph specify how terminal assemblies are passed between bins. This system has one unique terminal assembly, shown in the bin in stage 3.

1.4 Two Handed Assembly Model with Prebuilt Assemblies

The Staged Tile Assembly Model provides interesting functionality in that in stages after the first, the bins within those stages act as 2HAM systems that start with prebuilt assemblies instead of just singular tiles. It is interesting to think of a middle ground between multistage

Staged Tile Assembly and the 2HAM. One approach to this is consider a 2HAM model in which prebuilt assemblies area allowed. We refer to the 2HAM generalization which allows for the specification of prebuilt assemblies as P2HAM. Note that the difference between P2HAM and even 2 stage tile assembly is significant, as even seemingly simple problems such as producibility in two stage tile assembly are undecidable [29]. Additionally, due to the nondeterministic nature of attachments, a bin in the second stage can start with exponentially many unique assemblies relative to the size of the defined system. P2HAM limits this, as the specification of prebuilt assemblies is a factor in the size of the input.

1.5 Covert Computation

Self assembly systems can be considered a model of computation, In [31], Winfree showed that the aTAM was Turing Universal, i.e., capable of any computation that a Turing machine is capable of, and therefore capable of any physically realizable form of computation. In [7], Cantu et. al. introduced the notion of *covert computation*. This is the process in which a self assembly system can be used to compute some function, and the result of this function can be interpreted without revealing what the input to that function was. This is not a trivial task in self assembly, as the computational process of self assembly system is usually easily seen by observing the state of the system at the end of the self assembly process.

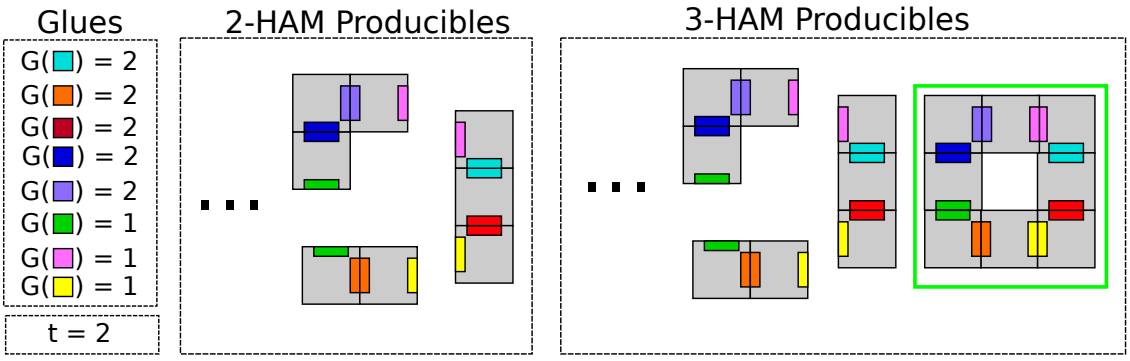


Figure 1.2: Example of the difference between 2HAM and 3HAM. In the 2HAM system the 3 producibles shown can not attach to each other. Each pair of two assemblies does not meet the threshold of 2. In the 3HAM system the 3 assemblies coming together in one step allows for the production of a stable assembly.

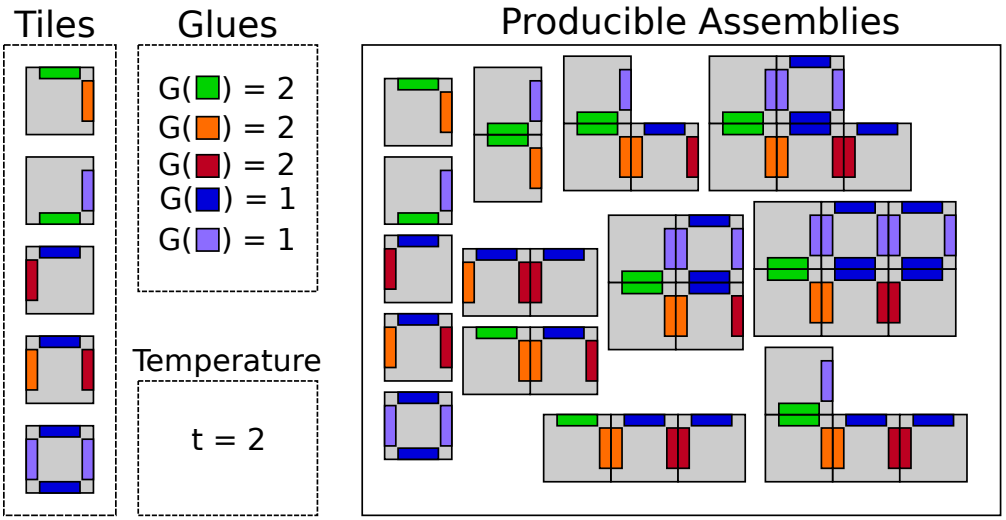


Figure 1.3: Example 2HAM system

1.6 Common Problems in Self Assembly

We define problems that are common to many models of self assembly, we define them in a general sense, due to the fact that they have different formal definitions depending on the model. For model specific definitions see Sections 2.1, 3.1, and 4.1.

Producibility

Input: A self assembly system T, an assembly A

Question: Is A a producible assembly in the system T?

Unique Assembly Verification (UAV)

Input: A self assembly system T, an assembly A

Question: Is A the uniquely assembled by T?

Unique Shape Verification (USV)

Input: A self assembly system T, a shape S

Question: Does every terminal assembly in T have shape S?

The producibility problem simply asks whether a system can produce a given assembly.

The Unique Assembly Verification problem is asking whether a system behaves as expected. Will the continual assembly process of some system uniquely yield one assembly, no matter the choice of assembly steps? The Unique Shape Verification problem asks a similar question, but no longer cares about the individual tiles that make up an assembly, but instead asks if all assemblies that a system yields are of one specific shape. The unique rules that the different self assembly models follow allows for these problems to have vastly different computational complexities when asking about different models.

1.7 Our Contributions

In this paper we prove several results related to the complexity of verification problems in the previously described models. Table 1.1 shows an overview of previous relevant results as well as results proven in this paper.

For the case of the P2HAM, we showed that the producibility problem is NP-Complete, while UAV problem is coNP^{NP}-Complete. These is true even when limited to $\mathcal{O}(1)$ size assemblies. This is a substantial increase in complexity from the 2HAM, in which these problems are in P, and coNP-Complete, respectively.

In the Staged Tile Assembly Model we prove that the problem of Unique Assembly Verification, which in this case is asks if a given assembly is uniquely produced by every bin in the *last* stage of a system, is PSPACE-Complete. We achieve this by utilizing a recently introduced technique, covert computation. We first showed that Staged Tile Assembly is capable of covert computation, and built on this to prove the main result of UAV complexity.

In the k-Handed Assembly Model, we show that the producibility problem admits a polynomial time algorithm. We then prove hardness for two formulations for the Unique Assembly Verification problem. We prove that UAV problem is coNEXP-Complete if the given integer k is encoded in binary, and the problem is coNP-Complete if the integer k is encoded in unary. These results show a vast difference in the complexity of verification problems between these models.

Problem	**Model**	**Results**	**Thm.**
Producibility	2HAM	P	[13]
UAV	2HAM	NP-Complete	[5]
Producibility	P2HAM	NP-Complete	Thm. 2.2.1
UAV	P2HAM	coNP^{NP}-complete	Thm. 2.3.3
UAV	Staged	PSPACE-Complete	Thm. 3.3.6
Producibility	k-HAM	P	Thm. 4.2.3
UAV	k-HAM (Binary)*	NEXP-Complete	Thm. 4.3.3
UAV	k-HAM (Unary)	coNP-Complete	Cor. 4.3.4

Table 1.1: Related and new results for verification problems. *The Binary and Unary specifiers denote how the integer k would be encoded for a provided instance of the problem.

CHAPTER II

TWO HANDED ASSEMBLY MODEL WITH PREBUILT ASSEMBLIES

In this section we focus on a specific generalization of the standard 2-handed tile self-assembly model (2HAM) in which we permit initial assemblies to consist of *prebuilt* assemblies of more than one tile. The motivation for studying such a generalization is strong. First, some of the most successful implementations of algorithmic DNA self-assembly utilize a combination of singleton DNA tiles mixed with larger prebuilt assemblies. For example, the experimental implementations of DNA tile counters [16] and the 21 DNA tile circuits implemented in [34] both utilize a combination of single tiles seeded with a larger prebuilt DNA origami structure encoding the program input. In [16], they additionally include a mix of singleton and prebuilt domino assemblies among the system's tile set. Second, the inclusion of prebuilt shapes of different geometries and sizes allows for the potential application of *steric hindrance*, in which geometric blocking of potential attachments is used to control algorithmic growth, as seen in the theoretical works of [11, 17, 18], and the experimental works of [14, 32]. These examples suggest that consideration of shapes more general than uniform single squares has the promise to allow for improved computational power and efficiency of self-assembled systems.

Given the importance of understanding self-assembly with prebuilt initial assemblies, we consider the complexity of two fundamental computational questions related to verifying the correctness of such systems. First is the *Producibility* problem, which asks if a given tile system can build/produce a given assembly. The second is the *Unique Assembly Verification* (UAV) problem, which asks if a given tile system *uniquely* produces a given assembly, i.e., produces the

assembly and nothing else provided sufficient assembly time. For the producibility problem, the 2HAM with just single-tile initial assemblies has a polynomial time solution [13], whereas we show NP-completeness when prebuilt assemblies are permitted. In the case of the UAV problem with singleton tile initial assemblies, the problem resides in coNP [6] for a constant-bounded temperature threshold and is coNP-complete for larger temperature thresholds [28], whereas we show coNP^{NP}-completeness with prebuilt assemblies. In both scenarios, our hardness results hold even for prebuilt assemblies of a bounded $\mathscr{O}(1)$ size and $\mathscr{O}(1)$-bounded temperature thresholds.

2.1 Preliminaries

In this section we overview the related definitions related to the two-handed self-assembly model and the verification problems under consideration.

Tiles. A *tile* is a non-rotatable unit square with each edge labeled with a *glue* from a set Σ. Each pair of glues $g_1, g_2 \in \Sigma$ has a non-negative integer *strength* $\text{str}(g_1, g_2)$.

Configurations, bond graphs, and stability. A *configuration* is a partial function $A : \mathbb{Z}^2 \to T$ for some set of tiles T, i.e. an arrangement of tiles on a square grid. For a given configuration A, define the *bond graph* G_A to be the weighted grid graph in which each element of $\text{dom}(A)$ is a vertex, and the weight of the edge between a pair of tiles is equal to the strength of the coincident glue pair. A configuration is said to be τ-*stable* for positive integer τ if every edge cut of G_A has strength at least τ, and is τ-*unstable* otherwise.

Assemblies. For a configuration A and vector $\vec{u} = \langle u_x, u_y \rangle$ with $u_x, u_y \in \mathbb{Z}^2$, $A + \vec{u}$ denotes the configuration $A \circ f$, where $f(x,y) = (x + u_x, y + u_y)$. For two configurations A and B, B is a *translation* of A, written $B \simeq A$, provided that $B = A + \vec{u}$ for some vector $\vec{u}$. For a configuration A, the *assembly* of A is the set $\tilde{A} = \{B : B \simeq A\}$. An assembly $\tilde{A}$ is a *subassembly* of an assembly $\tilde{B}$, denoted $\tilde{A} \sqsubseteq \tilde{B}$, provided that there exists an $A \in \tilde{A}$ and $B \in \tilde{B}$ such that $A \subseteq B$. An assembly is τ-*stable* provided the configurations it contains are τ-stable. Assemblies $\tilde{A}$ and $\tilde{B}$ are τ-*combinable*

into an assembly $\tilde{C}$ provided there exist $A \in \tilde{A}$, $B \in \tilde{B}$, and $C \in \tilde{C}$ such that $A \cup B = C$, $A \cap B = \varnothing$, and $\tilde{C}$ is τ-stable.

Two-handed assembly. A Two-handed assembly system is an ordered tuple (S, τ) where S is a set of *initial* assemblies and τ is a positive integer parameter called the *temperature*. Each assembly in S must be τ-stable. For a system (S, τ), the set of *producible* assemblies $P'_{(S,\tau)}$ is defined recursively as follows:

1. $S \subseteq P'_{(S,\tau)}$.
2. If $A, B \in P'_{(S,\tau)}$ are τ-combinable into C, then $C \in P'_{(S,\tau)}$.

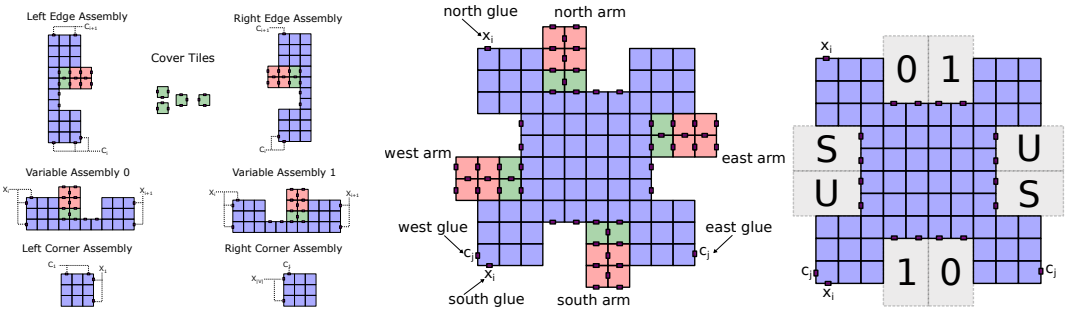
A producible assembly is *terminal* provided it is not τ-combinable with any other producible assembly, and the set of all terminal assemblies of a system (S, τ) is denoted $P_{(S,\tau)}$. Intuitively, $P'_{(S,\tau)}$ represents the set of all possible assemblies that can self-assemble from the initial set S, whereas $P_{(S,\tau)}$ represents only the set of assemblies that cannot grow any further. An assembly A is *uniquely produced* if $P_{(S,\tau)} = \{A\}$ and for each $B \in P'_{(S,\tau)}$ $B \sqsubseteq A$.

Definition 2.1.1 (Producibility Problem)**.** Given a P2HAM system $\Gamma = (S, \tau)$ and an assembly A, is A a producible assembly of Γ?

Definition 2.1.2 (Unique Assembly Verification Problem (UAV))**.** Given a P2HAM system $\Gamma = (S, \tau)$ and an assembly A, is A uniquely produced by Γ?

2.2 Producibility Hardness

Overview In this section, we show that even verifying producibility in the 2-handed assembly model with prebuilt assemblies is NP-complete with $\tau = 2$. We reduce from 3SAT, which asks whether a given 3CNF formula ϕ is satisfiable. Let $|V|$ and $|C|$ be the number of variables and clauses in ϕ, respectively. The reduction creates an instance of producibility (Γ, A) with $\tau = 2$, such that Γ produces the target assembly A iff ϕ is satisfiable. The target assembly is a rectangle shown and described in Figure 2.4a. The assemblies in the reduction can be divided into two



(a) (b) (c)

Figure 2.1: (a) Edge Assemblies used to construct the frame of the assembly. For each clause we include a left and right edge assembly. For each variable we include two variable assemblies representing 0 and 1. We include a single left corner assembly along with a right corner assembly. The filler tiles are used to fill in holes between attached macroblocks. (b) A single macro block $m_{i,j}(0,U,U)$ with outer glues labeled. The north and south glues connect to other macro blocks that represent the same variable. The east and west glues connects to other macroblocks that represent the same clause. (c) Arm position labels on a macroblock. Opposite sides have complementary values to allow for attachment.

groups: edge assemblies (Figure 2.1a) and macroblocks (Figure 2.1b). There are two variable assemblies for each bit that each correspond to an assignment of 0 or 1 based on the position of the 3×2 *arm* section of the assembly (green and red tiles in the figures). As shown in Figure 2.1c, each arm position represents a certain value. For vertical arms, the position represents either 0 or 1. For horizontal arms, their position is either U or S, meaning unsatisfied or satisfied.

These assemblies will combine to form an assembly for each possible assignment to ϕ (Figure 2.2a). We include a unique left edge assembly for each clause whose arm is always in the top position representing that the clause is currently not yet satisfied. A left edge assembly can attach to an assembly that encodes an assignment to ϕ. This starts to form an L shaped frame assembly as shown in Figure 2.2b. Macroblocks may attach to this frame if it has complementary arms to the currently growing frame assembly. As shown in Figure 2.2c, a macroblock can attach using two strength-1 glues on its east and south sides. If the arms have complimentary positions the attachment will be able to take place. However, if the arms overlaps, the macroblock is geometrically blocked from attaching, and thus not allowed.

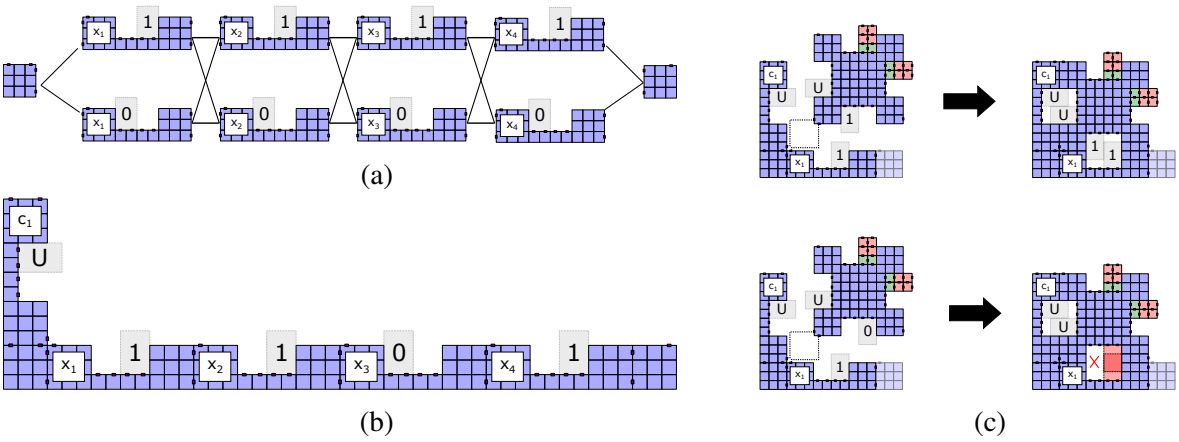


Figure 2.2: (a) Variable gadgets can nondeterministically build a frame assembly for each possible assignment to ϕ. (b) Example of the assembly made for assignment 1101 with a single left edge assembly attached. (c) If the macroblock has complimentary arm positions it will be able to attach to the frame assembly. If the macroblocks do not have matching arm positions the attachment will be geometrically blocked and cannot occur.

The final challenge for designing this reduction is there must exist a single assembly that is produced regardless of the satisfying assignment. This means we must hide the values passed between macroblocks. We do this by including a certain subset of the tiles which will fill any spaces left between macroblocks.

2.2.1 Macroblocks

A single macroblock can be seen in Figure 2.1b and has two parts: the body that contains glues to allow attachment (blue), and four arms which encode ϕ (green/red). Each arm on the macroblock encodes a single bit of information by being in one of two positions. We call these positions "0" and "1" for the north/south arms and "U" and "S" (unsatisfied/satisfied) for the east/west arms (Figure 2.1c).

We denote a macroblock representing a variable/clause pair (v_i, c_j) by its glues and arm positions as $m_{i,j}(b, w, e)$ where $b \in \{0,1\}$ is the position of the north/south arm, $w \in \{U, S\}$ is the position of the west arm, and $e \in \{U, S\}$ is the position of the east arm. Each macroblock has a single strength-1 glue on each side (macroblocks representing the last clause do not contain glues

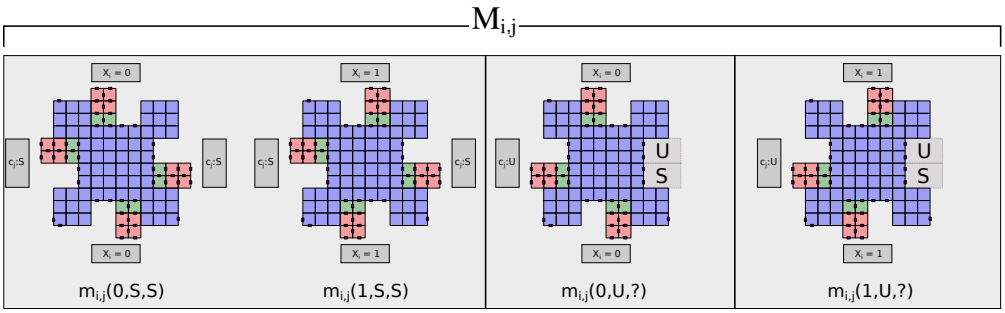


Figure 2.3: Possible Macroblocks that make up $\mathscr{M}_{i,j}$. Arm positions represent the value assigned to x_i and whether or not c_j has been satisfied. There will always be 4 macroblocks in each set. The left pair of macroblocks are always included and will attach if a clause is already satisfied. The remaining macroblocks attach if the clause is not yet satisfied, and their arm positions depend on ϕ. If the positive literal x_i is in c_j, $m_{i,j}(1,U,S) \in \mathscr{M}_{i,j}$, otherwise $m_{i,j}(1,U,U) \in \mathscr{M}_{i,j}$. If the negative literal $\bar{x}_i$ is in c_j, $m_{i,j}(0,U,S) \in \mathscr{M}_{i,j}$, otherwise $m_{i,j}(0,U,U) \in \mathscr{M}_{i,j}$.

or arms on their north side). The glues indicate which variable and clause pair this macroblock represents. The north and south glues relate to the variable, and the east and west glues relate to the clause. The south and west glues allow for cooperative attachment to an assembly that already contains macroblocks $m_{i-1,j}$ and $m_{i,j-1}$. The north and east glues allow for attachment of the next macroblocks.

Each variable/clause pair (v_i, c_j) has a set $\mathscr{M}_{i,j}$ of four macroblocks associated with it (shown in Figure 2.3). The exact macroblocks that are included depends on whether x_i or $\bar{x}_i$ is present in the j^{th} clause. The macroblocks $m_{i,j}(0,S,S)$ and $m_{i,j}(1,S,S)$ are always included since the assignment of a variable can not change a clause from being satisfied to unsatisfied. If the positive literal v_i appears in c_j, then we include the macroblocks $m_{i,j}(1,U,S)$, or $m_{i,j}(1,U,U)$ if it does not. If the negative literal $\bar{v}_i$ appears is in c_j we include the macroblock $m_{i,j}(0,U,S)$, or $m_{i,j}(0,U,U)$ if it does not.

2.2.2 Computing Clauses

The left edge assembly starts with an arm in the U position. Macroblocks maintain this position (Figure 2.4b) until a macroblock attaches that satisfies the clause, and changes the arm to an S position (Figure 2.4c). Once a row of the assembly is complete (Figure 2.4d), if the horizon-

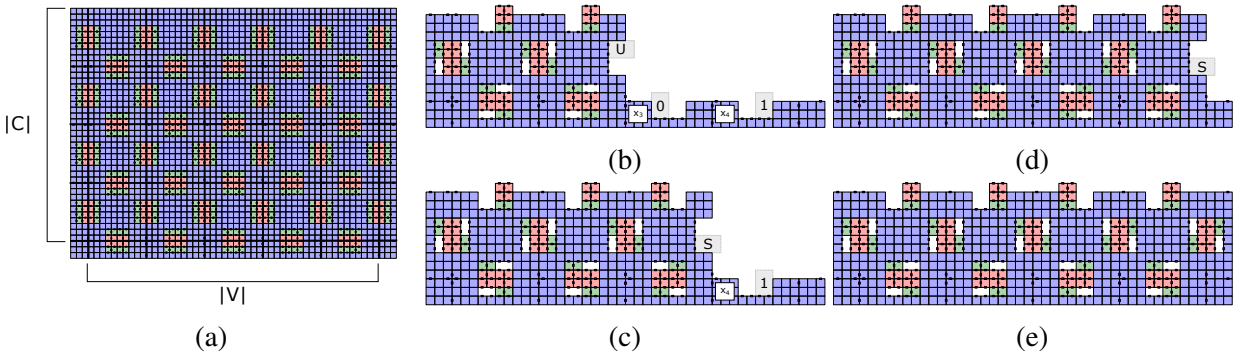


Figure 2.4: (a) Target assembly for producibility in the 2HAM with prebuilt assemblies. Target assembly is a $10|C|+3$ by $10|V|+6$ rectangle. Smaller rectangles between tiles denote strength-1 glues. Glues between blue tiles are not shown. Each blue tile shares a strength-2 glue with neighboring blue tiles. The exceptions are tiles separated by the thicker borders that do not share a glue unless shown. (b) A frame assembly with macroblocks attached. Here, $c_1 = \overline{x_1} \vee \overline{x_3} \vee x_4$. (c) Here $x_3 = 0$ satisfies the clause so this macroblock that attaches has its arm in the S position. (d) The macroblocks that attach after always have their arms in the S position. (e) The right edge assembly can attach to the last macroblock since its arm position is in on the S position completing the row.

tal arm is in the satisfied position, the right edge assembly can attach cooperatively and complete the row (Figure 2.4e). For a right edge assembly to attach, the clause must be satisfied and the assembly below it (either another right edge assembly or the right corner assembly) must attach as well. Thus, the right edge assembly only attaches if the clause it represents is satisfied.

Each row has multiple size-2 holes between macroblocks. Filler tiles cannot attach to macroblocks on their own due to the temperature of the system. However, once two macroblocks are attached, the tiles can cooperatively bind across the hole with strength-1 glues from each side (Figure 2.5b). As shown in Figure 2.5c, this hides the previously used arm positions. The exposed northern arms also continue to encode the input string so the next clause can be computed after the attachment of the next left edge assembly.

Theorem 2.2.1. The producibility problem in the 2HAM with prebuilt assemblies of size $\mathscr{O}(1)$ is NP-Complete with $\tau = 2$ and an initial assembly set containing $\mathscr{O}(|V||C|)$ distinct assemblies.

Proof. For membership in the class NP consider an assembly tree υ for a producible assembly A. To verify A is producible we may use υ as a certificate. If υ is a valid tree (each combination is legal) and each leaf is in the input set I, then A is producible.

To show NP-hardness we reduce from 3SAT. Given a formula ϕ in 3CNF form with $|V|$ variables and $|C|$ clauses. Our target assembly A is the rectangle described above made from macroblocks and edge assemblies with each of the spaces between arms completely filled with tiles. The input assembly set includes I, $|C|$ left edge assemblies with their arm in the unsatisfied position, and $|C|$ right edge assemblies in their satisfied position. For each variable, two input assemblies representing 0 and 1 assignments are included. The clauses are encoded by the selection of macroblock arm combinations. A set of 4 macroblocks are included in I for each variable and clause combination, so there are a total of $\mathcal{O}(|V||C|)$ macroblocks. This results in a total initial assembly set size of $\mathcal{O}(|V||C|)$, and each assembly is constant sized.

The starting assemblies, I, will grow into A if and only if ϕ is satisfiable. If ϕ is satisfiable by some assignment x, then the 'L' shaped frame assembly representing x will grow by attaching macroblocks. Since x satisfies ϕ each clause will eventually have its arm position changed to satisfied allowing for all the right edge assemblies to attach. The single tiles will then fill in the spaces to complete A. If A is producible then there exists some 'L' shaped frame assembled that grew into A. The only way this frame could have grown into A is if the position of the arms on the input assemblies represented a string x that satisfied each clause meaning ϕ is satisfiable. □

2.3 Unique Assembly Verification is coNP$^{\text{NP}}$-Complete

In this section we will show coNP$^{\text{NP}}$-Completeness of the Unique Assembly verification problem in the 2HAM with constant sized prebuilt assemblies. We start by proving UAV is in the class of problems solvable by a nondeterministic algorithm with access to an oracle for a problem in the class NP. We then show hardness by reducing from $\forall\exists$SAT. This is an extension of the

reduction shown in the previous section, on top of this we utilize similar techniques used in previous reductions in the 2HAM and Tile Automata [29, 4].

2.3.1 Membership

Lemma 2.3.1. The Unique Assembly Verification problem in the 2HAM with prebuilt assemblies is in coNP$^{\text{NP}}$.

Proof. Given an instance (Γ, A), refer to a "rogue assembly" R as a producible assembly in the system Γ that is either (1) not a subassembly of the target assembly A, $R \not\sqsubseteq A$, or (2) a strict subassembly of A and terminal, i.e., $R \sqsubset A$ and $R \in P_{(S,\tau)}$. The following nondeterministic algorithm solves UAV.

1. Nondeterministically build an assembly B of size $|B| \leq 2|A|$.
2. If B is a rogue assembly, reject.

It suffices to check all assemblies B up to size $2|A|$ since any assembly of size $> 2|A|$ must have been built from at least one other assembly B', s.t. $|A| < |B'| \leq 2|A|$. B' is a rogue assembly itself and will be accounted for in a different branch of the computation. It remains to be checked whether B is a rogue assembly. The first condition can be verified in polynomial time by checking if B is a subassembly of A. The second condition can be checked using an NP oracle that answers the following: "Does there exists an assembly C, $|C| \leq |A|$, such that C can attach to B?". This problem can be solved by an NP machine that nondeterministically builds an assembly C up to size A and attempts to attach it to B. If any C can attach to B, B is not terminal. If any branch finds a rogue assembly, the co-nondeterministic machine will reject. □

2.3.2 Reduction Overview

Definition 2.3.2 ($\forall\exists$SAT)**.** Given an n-bit Boolean formula $\phi(x_1, x_2 \ldots x_n)$ with the inputs divided into two sets X and Y, for every assignment to X, does there exist an assignment to Y such that $\phi(X,Y) = 1$?

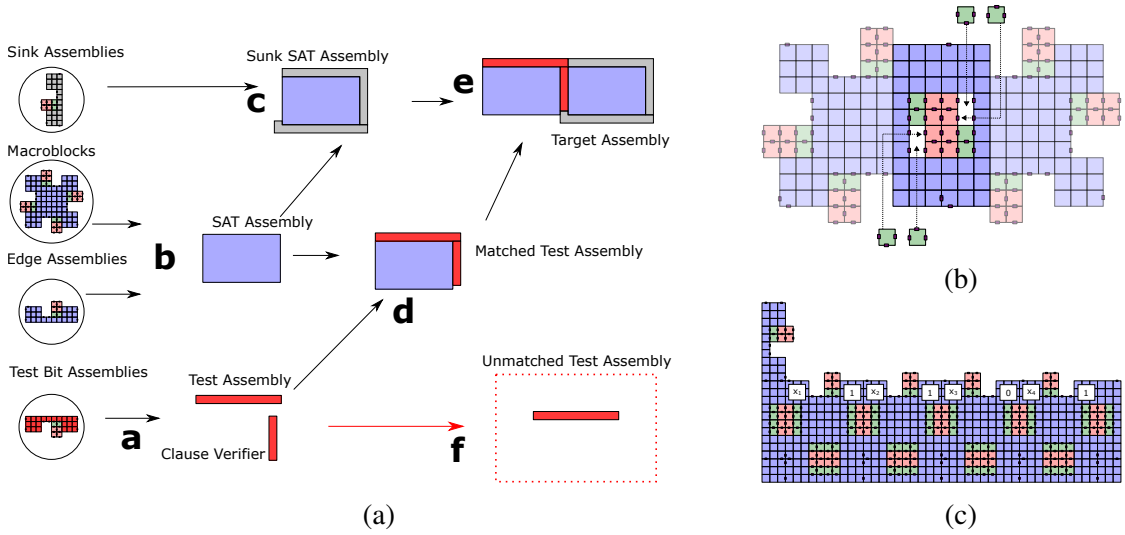
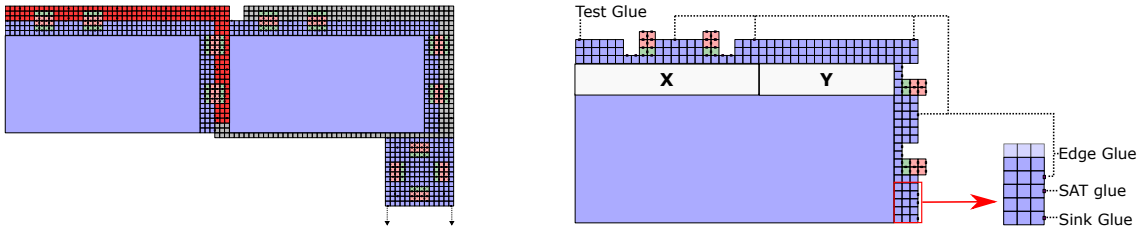


Figure 2.5: (a.a) Test bit assemblies come together to build a test assembly for all possible assignments of the variables in X. Clause Verifier assemblies may attach to SAT assemblies that satisfied all clauses. (a.b) Macroblocks and edge assemblies from the previous reduction are used to create SAT assemblies that evaluate the formula for every assignment of all the variables. (a.c) The sink assemblies begin attaching to SAT assemblies, ensuring they all grow into the target assembly. (a.d) A test assembly will attach to a SAT assembly that satisfies the formula and has matching assignments to the variables in X. (a.e) Matched test assemblies and sunk SAT assemblies attach to each other forming the target assembly. (a.f) Any test assembly that does not find a SAT assembly to attach to is unmatched and terminal. If any unmatched test assemblies exist, the instance of UAV is false. (b) Once two macroblocks attach, the green filler tiles are able to cooperatively attach using one glue on the macroblock, and the other glue from the red tiles of the arms from the other macroblock. The filling process hides the information that was passed. (c) Another left edge assembly may attach above the first. Since the north arms of macroblocks encode the variable assignment, the second clause may be computed in the same way as the first.

We show this problem is coNP$^{\text{NP}}$-hard by reducing from $\forall\exists$SAT. An overview of the important assemblies and processes are shown in Figure 2.5a. The same construction used in the previous reduction is used to create exponentially many 'SAT assemblies', each of which evaluates the Boolean formula on one of its input assignments. A SAT assembly is shown in Figure 2.6b. We do not include right edge assemblies so SAT assemblies that have finished computing will have exposed arms on their right side denoting whether or not each clause was satisfied. The assembly has exposed arms to the north above variables in X that represent their assigned values. Variables in Y will still have no arms on their north side. We construct a test assembly (Figure



(a) (b)

Figure 2.6: (a) Target Assembly for UAV. Assembly can be divided into three parts, Test assembly with satisfied SAT assembly, Sunk SAT assembly, and a column of clean up frames (b) SAT assembly. Built using the previous reduction with additional northern arms for the variables in X. Also we do not add right edge assemblies an arm is exposed which shows whether a clause has been satisfied.

2.7a) for each possible assignment to the variables in X (Figure 2.8a) along with a *clause verifier* assembly. The clause verifier assemblies are made of right edge assemblies with their arms in the S position. Each test assembly can attach to any SAT assembly with matching assignments to X (complimentary arm positions) if the clause verifier has already attached. Other assemblies are included that 'sink' all assemblies to the target except for test assemblies that did not attach to a SAT assembly. Test assemblies may build into the target assembly if and only if they find a matching SAT assembly. The key question about the system is "For all test assemblies, does there exist a compatible SAT assembly where all clauses evaluated to true?"

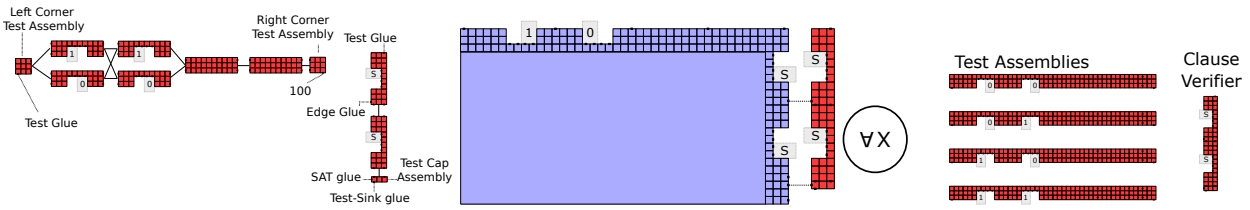
This system uniquely constructs the target assembly if and only if the instance of $\forall\exists$SAT is true. If the instance is false, there exists an assignment to the variables in X where no satisfying assignment of Y exists. In this case, the test assembly representing that assignment will be terminal, which means the system does not uniquely produce the target assembly.

SAT Assembly We use the assemblies from the previous reduction to compute all satisfying assignments to ϕ. We use the same macroblocks, input assemblies, and left edge assemblies, however, we do not include any right edge assemblies or the right corner assembly. A frame assembly is constructed for each assignment to ϕ. The assembly process then computes whether or not the assignment satisfies the clauses in the same way by attaching macroblocks with matching

geometry. In this construction we mark the assignment to variables in X by including northern arms to the top most macroblocks for those variables instead of omitting them as in the previous construction. This results in a final assembly that has its assignment to X and the computed value of each clause being encoded in the exposed arm positions.

The assembly can be seen in Figure 2.6b with glues labeled. The exposed glues all have strength-2. In the bottom right corner of the assembly, the last variable assembly has two glues. The bottom glue is called the sink glue and this is one of the glues the sink assembly uses to attach to a SAT assembly. The glue above it, called the SAT glue, is used by both the left sink base assembly and the test assembly. The next glue appears on each macroblock with an exposed arm, and the northern side of the rightmost macroblock. This glue allows for sink assemblies to attach and cover exposed arms to reach the target assembly. The test glue is the last glue on this assembly and appears in the top right corner. The test assembly uses this glue with the SAT glue to attach to a SAT assembly with matching geometry.

Test Assembly A test assembly is constructed for each possible assignment to X starting from constant sized test bit assemblies shown in Figure 2.7a. For each variable in X, we include two test bit assemblies. For each each variable in Y, we include a blank test bit assembly, which is a 3×10 rectangle. This row is capped by left and right corner test assemblies. The left and right corner assemblies have a strength-1 glue. The clause verifier assembly is built using right edge assemblies. These assemblies all have their arms in the satisfied position. The north most assembly has a strength-1 glue on its left side to attach to SAT assemblies and another on its north side to allow test assemblies to attach. They connect downward to a 1×3 test cap assembly. The test cap has the strength-1 SAT glue on its left side to allow a test assembly to cooperatively bind to a SAT assembly with all arms in the satisfied position (Figure 2.7b). On its south side it has the test-sink glue which will be used to attach to a sink assembly cooperatively once a SAT assembly is found. This assembly process creates a test assembly for each possible assignment



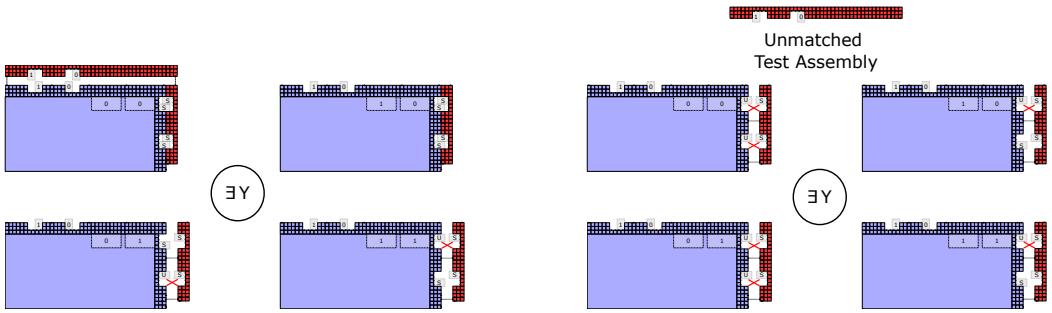
(a) (b) (c)

Figure 2.7: (a) Test bit assemblies nondeterministically construct a test assembly for each assignment to the variables in X. Right edge assemblies with arms in the satisfied positions are used to build the clause verifier. Only the north most edge assembly has an edge glue to allow a clause verifier to attach to a SAT assembly. A test assembly may attach to a SAT assembly with a clause checker attached using the test glues on their south side. The Test-Sink glue is used to cooperatively attach to a sink assembly once the test assembly is matched with a SAT assembly. (b) A clause verifier may attach to SAT assemblies that have their arms in the satisfied position. (c) Test Assemblies that are created for a X that contains 2 variables along with the single clause verifier.

to the variables in X. The example of test assemblies for an instance where $|X| = 2$ and $|Y| = 2$ can be seen in Figure 2.7c.

The test assembly has two exposed glues on opposite ends of the assembly. Thus, the assembly cannot attach to a SAT assembly until it is completely constructed and a clause verifier assembly has attached. A clause verifier assembly may only attach to SAT assembly with matching arm positions, i.e., SAT assemblies that satisfied all clauses. A test assembly may only attach to SAT assemblies with a clause verifier and that have the same assignment to X. A test assembly along with the four possible SAT assemblies it could attach to is shown in Figure 2.8a. A test assembly is terminal if there does not exist a SAT assembly with the same assignment to X that satisfies all clauses. A terminal test assembly can be seen in 2.8b. We call these terminal test assemblies *unmatched* test assemblies. Since we construct a SAT assembly for each possible assignment to the formula ϕ, if the test assembly representing a partial assignment x is terminal that means there does not exist a remaining assignment to the variables in Y that satisfies the formula, and causes the instance of UAV to be false.

Sink Since our goal is to design a system that uniquely constructs an assembly when the instance of $\forall\exists$SAT is true, we will *sink* assemblies representing a non-satisfying assignment



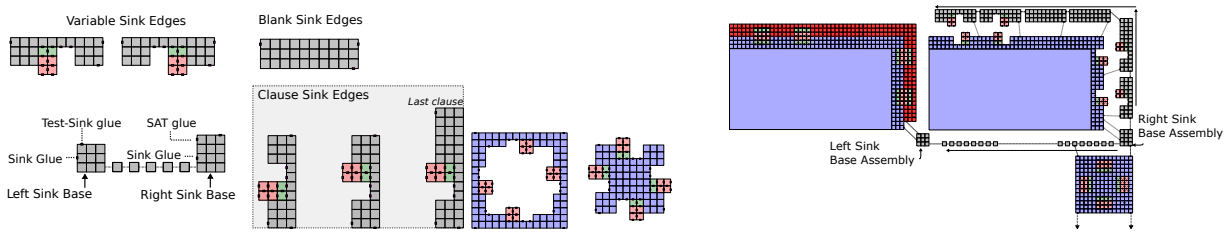
(a) (b)

Figure 2.8: (a) If there exists a satisfying assignment to Y when $X = 10$, the test assembly with those arms can attach to a SAT assembly. (b) If there does not exist a satisfying assignment, all the SAT assemblies will have at least one arm not in the S position that will geometrically block the test assembly's arm. This test assembly will be terminal so the answer to UAV will be no.

to the target assembly. This ensures that each assembly (besides unmatched test assemblies) must eventually grow into the target assembly thus sinking all other producible assemblies to our target. We do so via sink assemblies and macroblock frames. The sink assemblies are shown in Figure 2.9a. The first sink assembly, the right sink base assembly, is similar to the right corner assembly of the previous section but is of height 4. Sink base tiles are single tiles that may attach to the right base assembly on its left side bottom row, which eventually connects to the left base assembly.

The right sink base assembly attaches to any SAT assembly that has not attached to a test assembly using the sink and test-sink glues. We call the attachments that occur after this the *Sink Process*. During the Sink Process, sink edge assemblies attach cooperatively with the right sink base assembly and with the SAT assembly. This allows for the sink edge assemblies to attach one-by-one and 'cover' each exposed arm on the SAT assembly. The last sink edge (northern most) is slightly longer to allow for the horizontal sink edges to attach across the top of the assembly. The left sink base assembly cooperatively attaches to test assemblies that have already attached to SAT assemblies using the SAT glue on its right side and the test-sink glue on its north side.

The last assembly type that has not been accounted for in the final assembly are any unused macroblocks. In the previous reduction, any unused macroblocks are terminal since they



(a) (b) (c)

Figure 2.9: (a) The set of sink assemblies. The sink glue and test-sink glue are utilized for cooperative attachment. (b) For each macroblock we also include a frame so none of the macroblocks are terminal. These frames attach to each other in order at the bottom of the sink assembly. (c) The process of sinking all assemblies towards the target assembly.

are never used in the computation. In this reduction, we cannot have any other terminal assembly, so these must be included in our target assembly. We do this by adding frames to store the macroblocks. For each macroblock we include in our input set, we also include a clean up frame (Figure 2.9b). Any macroblock is now not terminal as it can attach to the clean up frame. These frames are enumerated and attach in order to the south side of the sink assembly in a single column (Figure 2.9c).

Theorem 2.3.3. The Unique Assembly Verification problem in the 2HAM with prebuilt assemblies of size $\mathscr{O}(1)$ is coNP$^{\text{NP}}$-complete with an initial assembly set size of $\mathscr{O}(|V||C|)$ and $\tau = 2$.

Proof. We show membership in Lemma 2.3.1. Given an instance of $\forall\exists$SAT with a formula $\phi(x_1, x_2, \ldots, x_n)$ we create a 2HAM system S that uniquely assembles a target assembly S if and only if the instance of $\forall\exists$SAT is true. If the instance of $\forall\exists$SAT is false then S will produce a test assembly that is terminal.

The construction of SAT assemblies begins with combinations of variable and edge assemblies to produce an L shaped frame assembly for each possible assignment to ϕ. The output of ϕ on each assignment is then computed by the attachment of macroblocks that encode the clauses of ϕ, and producing a SAT assembly. The SAT assemblies expose arms representing the assignment of the variables in X, as well as arms that represent whether each clause has been

satisfied. From the included prebuilt assemblies, a test assembly is produced for each possible assignment to X. Due to their arm positions, they may only attach to SAT assemblies that have a matching assignment to X and that represent an assignment that satisfies every clause.

Each assembly besides unmatched test assemblies will sink to the target assembly. Every prebuilt assembly that was designed for building a SAT assembly will be used in the construction of at least one SAT assembly, besides the macroblocks. In some cases it is possible for a macroblock to not be able to attach to any SAT assembly. To account for this this we include a macroblock frame for each macroblock to attach to, ensuring that no macroblock is terminal. The right sink base assembly may attach to any SAT assembly regardless of arm position so none of the SAT assemblies are terminal. Each sink assembly is used in the process of reaching the target assembly so none are terminal.

A test assembly is only terminal if there does not exist a SAT assembly with matching X arm positions that has each clause satisfied. This means for that assignment to X there does not exist an assignment to Y that satisfied ϕ, and the instance of $\forall\exists$SAT is false.

The new assemblies in this reduction only increase the number of assemblies by a constant factor. The test and sink assemblies only add $\mathcal{O}(|V|+|C|)$ to the input assembly size, and the added macroblock frames are equal to the number of macroblocks, which is constant. □

CHAPTER III

THE STAGED SELF ASSEMBLY MODEL

The *Staged Self-Assembly* model was designed as an extension to the standard hierarchical model of tile self assembly that mimics the abilities of scientists in the lab to control the assembly process by mixing test tubes. The additional features in this model allow for more efficient tile complexity, but increased complexity of certain verification problems.

We use the concept of *Covert Computation*, a requirement of a computational system stipulating that the input and computational history of the computation be hidden in the final output of the system, within the context of Staged Self-assembly, an extension to tile self-assembly that allows for basic operations such as mixing self-assembly batches over a sequence of distinct stages. We use this connection to resolve open questions regarding the complexity of the *Unique Assembly Verification* (UAV) problem within staged self-assembly- the problem of whether a given system uniquely produces a specific assembly. The importance of this work stems from the fundamental nature of the UAV problem, along with the natural and experimentally motivated Staged Self-Assembly model. Further, the novel approach by which our results are obtained, by way of designing Covert Computation systems in Staged Self-Assembly, may be of independent interest as it shows how to utilize Staged Self-Assembly to implement general purpose computing systems with strong guarantees that might be useful for cryptography or have applications for privacy within biomedical computation.

In this section, we introduce the concept of covert computation in the context of staged self-assembly for the purpose of establishing the complexity of unique assembly verification

within the model. First, we show that staged self-assembly is capable of covert computation even when limited to three stages. Next, we use this fact to show UAV is PSPACE-complete in staged self-assembly, resolving the open problem from [29]. Along the way, we improve on some results from [29]: we show that UAV is Π_2^p-hard with just three stages, improving on the previous hardness result requiring seven stages. We then generalize this result to show that for n stages, UAV is Π_{n-1}^p-hard, but yields a Π_{n+1}^p algorithm, leaving only a gap of two in levels between membership and hardness for this problem.

3.1 Preliminaries

We first provide definitions for the staged self-assembly model and covert computation.

Tiles. A *tile* is a non-rotatable unit square with each edge labeled with a *glue* from a set Σ. Each pair of glues $g_1, g_2 \in \Sigma$ has a non-negative integer *strength* $\text{str}(g_1, g_2)$.

Configurations, bond graphs, and stability. A *configuration* is a partial function $A : \mathbb{Z}^2 \to T$ for some set of tiles T, i.e. an arrangement of tiles on a square grid. For a given configuration A, define the *bond graph* G_A to be the weighted grid graph in which each element of $\text{dom}(A)$ is a vertex, and the weight of the edge between a pair of tiles is equal to the strength of the coincident glue pair. A configuration is said to be τ-*stable* for positive integer τ if every edge cut of G_A has strength at least τ, and is τ-*unstable* otherwise.

Assemblies. For a configuration A and vector $\vec{u} = \langle u_x, u_y \rangle$ with $u_x, u_y \in \mathbb{Z}^2$, $A + \vec{u}$ denotes the configuration $A \circ f$, where $f(x,y) = (x + u_x, y + u_y)$. For two configurations A and B, B is a *translation* of A, written $B \simeq A$, provided that $B = A + \vec{u}$ for some vector $\vec{u}$. For a configuration A, the *assembly* of A is the set $\tilde{A} = \{B : B \simeq A\}$. An assembly $\tilde{A}$ is a *subassembly* of an assembly $\tilde{B}$, denoted $\tilde{A} \sqsubseteq \tilde{B}$, provided that there exists an $A \in \tilde{A}$ and $B \in \tilde{B}$ such that $A \subseteq B$. An assembly is τ-*stable* provided the configurations it contains are τ-stable. Assemblies $\tilde{A}$ and $\tilde{B}$ are τ-*combinable* into an assembly $\tilde{C}$ provided there exist $A \in \tilde{A}$, $B \in \tilde{B}$, and $C \in \tilde{C}$ such that $A \cup B = C$, $A \cap B = \varnothing$, and $\tilde{C}$ is τ-stable.

Two-handed assembly and bins. We define the assembly process in terms of bins. A *bin* is an ordered tuple (S, τ) where S is a set of *initial* assemblies and τ is a positive integer parameter called the *temperature*. For a bin (S, τ), the set of *produced* assemblies $P'_{(S,\tau)}$ is defined recursively as follows:

1. $S \subseteq P'_{(S,\tau)}$.
2. If $A, B \in P'_{(S,\tau)}$ are τ-combinable into C, then $C \in P'_{(S,\tau)}$.

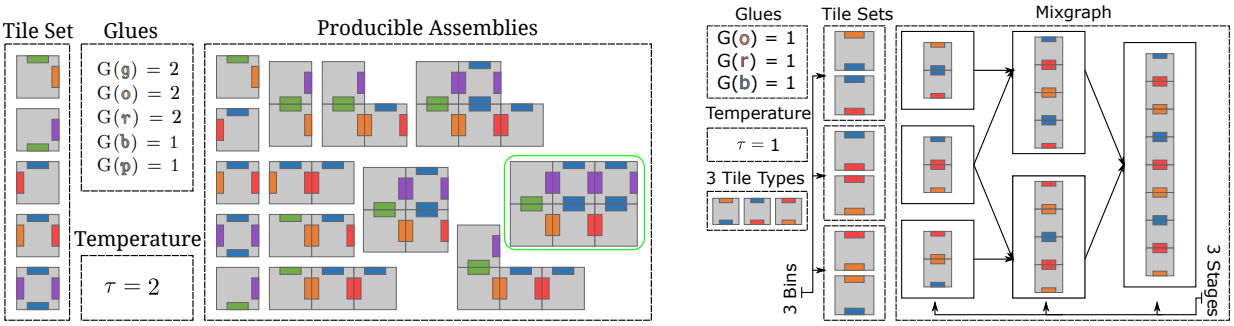
A produced assembly is *terminal* provided it is not τ-combinable with any other producible assembly, and the set of all terminal assemblies of a bin (S, τ) is denoted $P_{(S,\tau)}$. Intuitively, $P'_{(S,\tau)}$ represents the set of all possible assemblies that can self-assemble from the initial set S, whereas $P_{(S,\tau)}$ represents only the set of supertiles that cannot grow any further. The assemblies in $P_{(S,\tau)}$ are *uniquely produced* iff for each $x \in P'_{(S,\tau)}$ there exists a corresponding $y \in P_{(S,\tau)}$ such that $x \sqsubseteq y$. Thus unique production implies that every producible assembly can be repeatedly combined with others to form an assembly in $P_{(S,\tau)}$.

Staged assembly systems. An *r-stage b-bin mix graph* $M_{r,b}$ is an acyclic r-partite digraph consisting of rb vertices $m_{i,j}$ for $1 \leq i \leq r$ and $1 \leq j \leq b$, and edges of the form $(m_{i,j}, m_{i+1,j'})$ for some i, j, j'. A *staged assembly system* is a 3-tuple $\langle M_{r,b}, \{T_1, T_2, \ldots, T_b\}, \tau \rangle$ where $M_{r,b}$ is an r-stage b-bin mix graph, T_i is a set of tile types, and τ is an integer temperature parameter.

Given a staged assembly system, for each $1 \leq i \leq r$, $1 \leq j \leq b$, we define a corresponding bin $(R_{i,j}, \tau)$ where $R_{i,j}$ is defined as follows:

1. $R_{1,j} = T_j$ (this is a bin in the first stage);
2. For $i \geq 2$, $R_{i,j} = \Big(\bigcup_{k:\ (m_{i-1,k}, m_{i,j}) \in M_{r,b}} P_{(R_{(i-1,k)}, \tau)} \Big)$.

Thus, the j^{th} bin in stage 1 is provided with the initial tile set T_j, and each bin in any subsequent stage receives an initial set of assemblies consisting of the terminally produced assem-



(a) 2HAM Example

(b) Staged Self-Assembly Example

Figure 3.1: (a) A 2HAM example that uniquely builds a 2×3 rectangle. The top 4 tiles in the tile set all combine with strength-2 glues building the 'L' shape. The tile with blue and purple glues needs two tiles to cooperatively bind to the assembly with strength 2. All possible producibles are shown with the terminal assembly highlighted. (b) A simple staged self-assembly example. The system has 3 bins and 3 stages, as shown in the mixgraph. There are three tile types in our system that we assign to bins as desired. From each stage only the terminal assemblies are added to the next stage. The result of this system is the assembly shown in the bin in stage 3.

blies from a subset of the bins in the previous stage as indicated by the edges of the mix graph.[1] The *output* of the staged system is the union of all terminal assemblies from each of the bins in the final stage.[2] We say this set of output assemblies is *uniquely produced* if each bin in the staged system uniquely produces its respective set of terminal assemblies.

Definition 3.1.1 (Unique Assembly Verification Problem (UAV))**.** Given a Staged Tile Assembly system $\Gamma = (M_{r,b}, \{T_1, T_2, \ldots, T_b\}, \tau)$ and an assembly A, is A uniquely produced by Γ?

3.1.1 Covert Computation

Tile assembly computers were first defined in [8, 21] with respect to the aTAM. We provide formal definitions of both Tile Assembly Computers and Covert Computation with respect to the Staged Self-Assembly model.

[1] The original staged model [11] only considered $\mathcal{O}(1)$ distinct tile types, and thus for simplicity allowed tiles to be added at any stage. Since our systems may have super-constant tile complexity, we restrict tiles to only be added at the initial stage.

[2] This is a slight modification of the original staged model [11] in that the final stage may have multiple bins. However, all of our results apply to both variants of the model.

A Staged Tile Assembly Computer (STAC) for a function f consists of a staged self-assembly system, and a format for encoding the input into tiles sets and a format for reading the output from the terminal assembly. The input format is a specification for what set of tiles to add to a specific bin in the first stage. Each bit of the input must be mapped to one of two sets of tiles for the respective bit position: a tile set representing "0", or tile set representing "1". The input set for the entire string is the union of all these tile sets. Our staged self assembly system, with the set of tiles needed to build the input seed added in a designated bin, is our final system which performs the computation. The output of the computation is the terminal assembly the system assembles. To interpret what bit-string is represented by the assembly, a second *output* format specifies a pair of sub-assemblies and locations for each bit. An assembly that represents a bitstring is created by the union of each sub-assembly represented by each bit.

For a STAC to *covertly* compute f, the STAC must compute f and produce a unique assembly for each possible output of f. Thus, for all x such that $f(x) = y$, a covert STAC that computes f produces the same output assembly representing output y for each possible input x, making it impossible to determine which input value x was provided to the system.

Input Template. An n-bit input template over tile set T is a sequence of ordered pairs of tile sets over T: $I = (I_{0,0}, I_{0,1}), \ldots, (I_{n-1,0}, I_{n-1,1})$. For a given n-bit string $b = b_0, \ldots, b_{n-1}$ and n-bit input template I, the input tile set for b with respect to I is the set $I(b) = \bigcup_i I_{i,b_i}$.

Output Template. An n-bit output template over tile set T is a sequence of ordered pairs of configurations over T: $O = (C_{0,0}, C_{0,1}), \ldots, (C_{n-1,0}, C_{n-1,1})$. For a given n-bit string $x = x_0, \ldots, x_{n-1}$ and n-bit output template O, the *representation* of x with respect to O is $O(x)$ = the assembly of $\bigcup_i C_{i,x_i}$. A template is valid for a temperature parameter $\tau \in \mathbb{Z}^+$ if this union never contains overlaps for any choice of x, and is always τ-stable. An assembly $B \supseteq O(x)$, which contains $O(x)$ as a subassembly, is said to represent x as long as $O(d) \nsubseteq B$ for any $d \neq x$.

Function Computing Problem. A *staged tile assembly computer* (STAC) is an ordered triple $\Im = (\Gamma, I, O)$ where $\Gamma = (M, \{\varnothing, T_2, \ldots, T_i\}, \tau)$ is a staged self assembly system, I is an n-bit input template, and O is a k-bit output template. A STAC is said to compute function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^k$ if for any $x \in \mathbb{Z}_2^n$ and $y \in \mathbb{Z}_2^k$ such that $f(x) = y$, then the staged self assembly system $\Gamma_{\Im,x} = (M, \{I(x), T_2, \ldots, T_i\}, \tau)$ uniquely assembles a set of assemblies which all represent y with respect to template O.

Covert Computation. A STAC *covertly* computes a function $f(x) = y$ if 1) it computes f, and 2) for each y, there exists a unique assembly A_y such that for all x, where $f(x) = y$, the system $\Gamma_{\Im,x} = (M, \{I(x), T_1, \ldots, T_i\}, \tau)$ uniquely produces A_y. In other words, A_y is determined by y, and every x where $f(x) = y$ has the exact same final assembly.

3.2 Covert Computation in Staged Self-assembly

Here, we demonstrate covert computation in the staged assembly model. This construction creates a logic circuit using a 3-stage temperature-2 system with a number of bins polynomial in the size of the circuit. We consider only circuits made up of functionally universal NAND gates, but these techniques could be used to create any 2-input gate.

Figure 3.2b shows a basic overview of the mixgraph used for the covert computation implementation. The method requires three stages with a polynomial number of mixing bins.

- In the first stage, we assemble the components needed to perform the computation. These include an *Input Assembly*, which encodes the input to the function, *Gate Assemblies*, which act as individual gates and perform the computation via their attachment rules and geometry, and additional assemblies which are used to help "clean up" our circuit and covertly get the output.

- In stage two, the input assembly and gate assemblies are added to a single bin along with a test tile. The gate assemblies will begin to attach to the input assembly creating a *Circuit Assembly*. Once the computation is complete, the test tile can attach to the circuit assembly

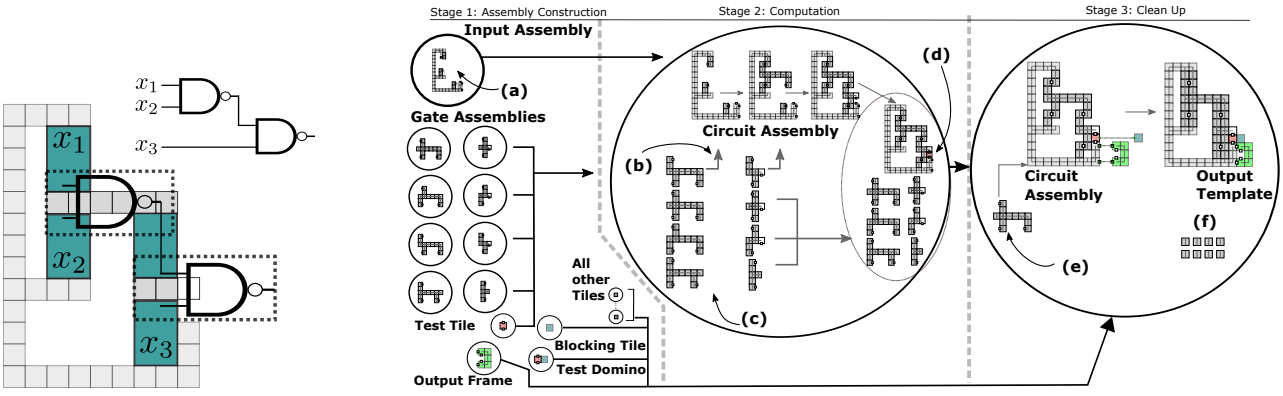
if and only if the output is true. The circuit assembly is terminal in this bin and will be passed to the final stage.

- The final stage adds additional assemblies to the bin along with most of the tile set as single tiles (not shown in figure). The additional assemblies read the output of the circuit and it grows into one of the output templates. The *Output Frame* searches for the test tile representing the output of the circuit. The single tiles fill in any spaces left in the circuit assembly that would show the computation history, thereby turning the assembly into the output template. This requires a linear number of additional bins in the first and second stage to store these single tiles while mixing takes place in other bins.

For our circuit assembly we implement Planar Logic Circuits with only NAND gates. An example circuit and an assembly showing how the gates are laid out are shown in Figure 3.2a. Wires are represented by 2×3 blocks of tiles shown in blue in the image. Input and Gate assemblies contain a subset of the tiles in each block we call *arms* which represent the values being passed along the wires. The input assembly is a comb-like structure that is designed so that each input bit reaches the gate it is used at (Figure 3.3a). For each NAND gate in the circuit we have 4 different assemblies, one for each possible input to the gate. A gate assembly can cooperatively bind to the input assembly if the variable values match. The gate assembly has a third arm that represents the output. This allows the next gate assembly to attach, which continues propagating until the computation is done and the circuit assembly is complete. We now cover the construction in detail by stage.

3.2.1 First Stage - Assembly Construction

Each bin in the first stage will individually create the assemblies that will come together in the next stage. For an n-input k-gate NAND logic circuit (considering crossovers as three XOR gates [8]), we have an input assembly, $4k$ gate assemblies, and a constant number of other assemblies that will be used in the final stage. Here we will describe the details of the individual



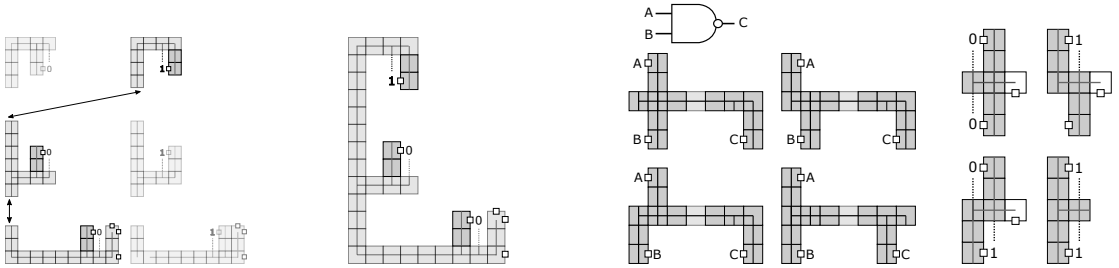
(a) (b)

Figure 3.2: (a) Simple 3-input logic circuit using 2 NAND gates, and the high-level abstraction of the circuit assembly showing the input variables and gates highlighted as blocks. Blue blocks are the sections of the assemblies we call *Arms* that function as wires in the systems. (b) (1) Our input assembly and gate assemblies are constructed in separate bins. (2) Gate assemblies attach to the input assembly forming a circuit assembly. (3) Unused gates are terminal in the second stage. (4) This circuit evaluates to true, so the test tile will be able to attach. (5) Gate assemblies in this stage grow into a circuit using single tiles. (6) Single tiles fill in open spots in the circuit assembly to hide the history. The additional assemblies are used to reach the output template.

assemblies created in addition to the *arms*, which function as wires in our system.

Input. For each bit of the input we have two possible input bit assemblies (Figure 3.3a). The value of the bit determines which tiles will be added to create that input bit assembly in the first stage. Figure 3.3a shows the selected assemblies that come together to form the *input assembly* shown in Figure 3.3b. Each subassembly has a domino which we call an 'arm' representing the corresponding bit value. The shape of these assemblies depends on the gates to which they input because the arm of the assembly must reach the location of the gate it inputs to. The last input bit assembly also contains an extra set of tiles that reach the final output gate with a strength-1 glue on its north end and two glues on it's east to allow for the test tile and output frame to attach.

Arms. We describe assemblies as having input or output *arms* which function as the wires of our circuit. Arms are vertical dominoes that represent bit values, with their location on the assembly representing the bit having a value 0 or 1 (Figure 3.4a). The output arm being in the left position represents a bit value of 0, with the right position representing 1. The locations

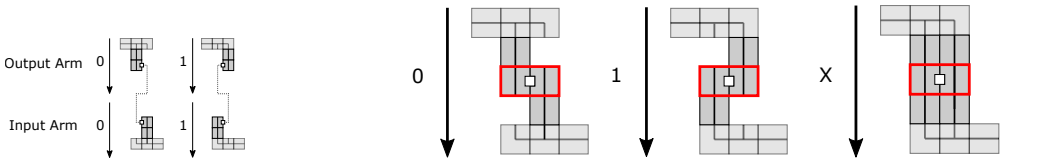


(a) (b) (c) (d)

Figure 3.3: (a) Possible input bit assemblies for a 3 bit function. Solid lines between tiles indicate a strength 2 glue between the tiles. Small boxes indicate a strength 1 glue. For each bit we select either the left or the right assembly based on the value of the bit and add those tiles to our input bin in the *First Stage*. Lighter tiles are not used. (b) The input assembly that is constructed in the *First Stage*. The last input bit assembly contains an extra column of tiles that reaches to where the output gate will be for cooperative attachment of the test tile. (c) 4 gate assemblies, one for each possible input combination of a NAND gate. Glues are labeled to match the wires of the NAND gate. (d) Output gates. True output gates contain a flag tile (white).

of input arms are complementary (right represents 0, left represents 1) to the output arms. These arms have a glue on the second tile on the inner side. An input arm will attach to an output arm to "read" the bit (Figure 3.4b) if they represent the same wire and the same value. This glue is a strength-1 glue, so the assembly must attach cooperatively elsewhere in the assembly. Another key feature of these arms is the ability to hide the information passed through by adding single tiles in a later stage. The spaces left by the attachment may be filled by single tiles which results in an assembly which looks like Figure 3.4c where the value passed cannot be read. This feature will be used in the final stage of our system.

Gates. For each gate we create four assemblies with each representing one of the valid input/output combinations of the desired logic gate. Each gate assembly has two input arms and one output arm. We encode the logic gate by placing the output arm in the column representing the output of the gate when input with the bits represented by the input arms (Figure 3.3c). This assembly has strength-1 glues on each of its arms. The shape of each gate is dependent on the layout of the circuit since the output arm needs to reach to the next gate. In the case a gate has a fan out (outputs to multiple gates) a gate assembly may have multiple output arms which share



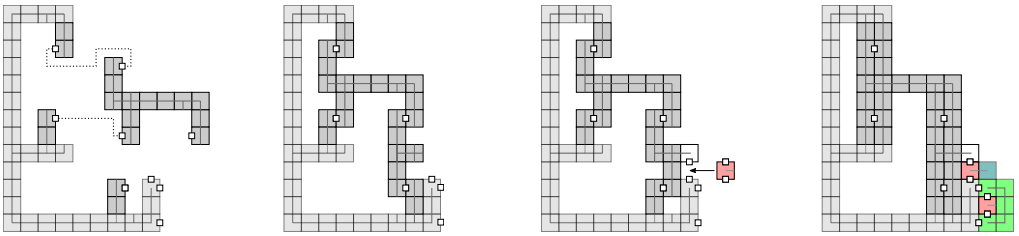
(a) (b) (c)

Figure 3.4: (a) Information being passed along a wire is represented by the position of a domino called an arm. Output arms represent a signal of "1" of "0" by being in the left or right position, respectively. Input arms read bit values and have complimentary arm placement to allow for attachment. (b) Information is passed by attachment. Another assembly may attach if the arms have matching glues (they represent the same wire) and they have complementary arms (represent the same bit value). (c) In the final stage we add additional tiles to hide the information that was passed along a wire.

arm position. We will refer to the final gate of the circuit as the *output gate*. It does not contain an output arm but instead contains a flag tile to represent an output of false, or no flag tile to represent an output of true which can be seen in Figure 3.3d. The flag tile also contains a strength-1 glue on it's south edge which allows for the test tile to attach.

3.2.2 Second Stage - Computation

In the second stage there is a single bin where the circuit assembly is created. In this stage the input assembly and the gate assemblies are mixed together to compute the encoded circuit. The computation starts by attaching gates to the input assembly to begin to build the circuit assembly. Once both inputs to a gate are present on the circuit assembly, the next gate assembly can cooperatively attach to the circuit assembly since each arm has an attachment strength of 1 as seen in Figure 3.5a. In this stage we also add the test tile. If the output of the circuit is true, the flag tile can attach as in Figure 3.5c. If the output is false, the terminal circuit assembly can be seen in Figure 3.5b. The test assembly is not able to attach to the circuit assembly in this case and will be terminal. We note that at this point it is possible to read the output of the circuit by checking the terminal assemblies, however the computation history can still be read so the covert computation is not complete and we need an additional stage.



(a) (b) (c) (d)

Figure 3.5: (a) A NAND gate assembly representing input: “10” and output: “1” attaching to the input assembly in the *Second Stage*. (b) False Output Gates which do not contain a flag tile can attach to the circuit if the output is false. This assembly is terminal in the second bin. (c) True Output Gates have an additional flag tile (white) that allows for the test tile (red) to attach cooperatively to the input assembly and the True Output Gate. (d) Single tiles fill in the spaces left by the arms and the output frame attaches forming our target assembly. If there is a flag tile in the output frame the output of this circuit is true. Otherwise, the output is false.

Utility Gates. In order to implement general circuits we need to handle gates with more than one output and also be able to cross wires. Gates with a fan-out (outputs to more than one place) contain multiple output arms (Figure 3.6a). Non-monotone circuits can be created with crossover gates (Figure 3.6b). These gates have two input arms and two output arms. The bit of the upper input arm is represented by the lower output arm and the same for the other two arms.

3.2.3 Third Stage - Clean Up

In the third stage we hide the computational history and get the output of the computation. The output template (Full Circuit Assembly) is shown in Figure 3.5d, which is a circuit assembly with all open spaces in its arm filled in (computation history is hidden) and the additional assemblies attached.

The additional assemblies are the Output Frame, which the test tile may attach to, the test domino, which attaches to a circuit assembly but not to the output frame, the blocking tile, which turns a test tile into a test domino, and all single tiles used in the circuit assembly other than the test tile. The difference between the true and false output templates is the inclusion/exclusion of the test tile within the Output Frame. The last stage contains a single bin. The inputs to this bin are the previous computation bin from stage two, and bins from stage one containing the output

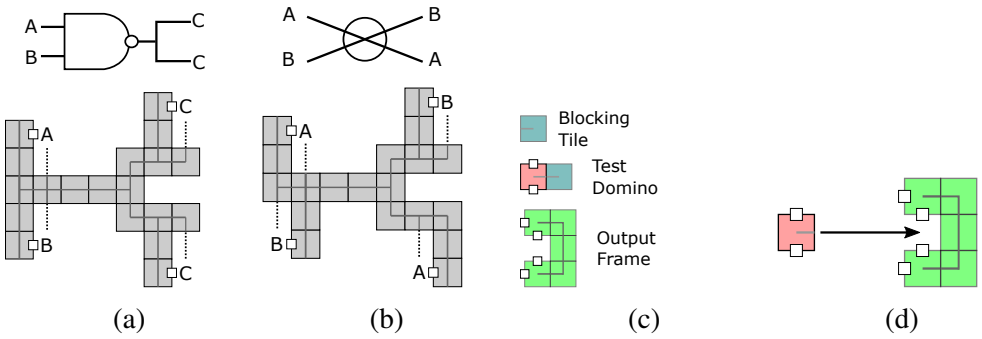
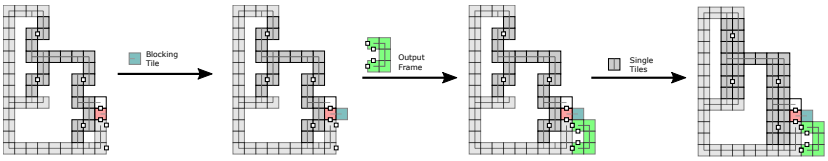


Figure 3.6: (a) A gate with a fan out (two outputs). (b) A crossover gate. (c) The assemblies added in the final stage to read the output and hide computation. The blocking tile attaches to test tiles that are a part of false circuits in order to reach the final assembly. The test domino is used to fill in true circuits that are passed into the third stage so they will build the final circuit assembly. The Output frame attaches to the edge of the circuit assembly and will grab the test tile if it is present. (d) In the *Third Stage* the output frame (green) is added. If the test tile (red) is present in this stage it can attach to the output frame.

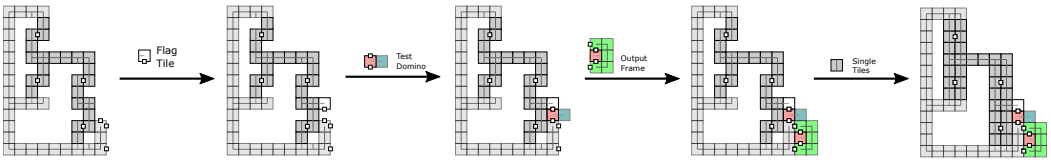
frame, blocking tile, and test domino (Figure 3.6c). We also include all single tiles of the circuit assembly.

The process of a circuit assembly which was terminal in the second stage growing into our output template can be seen in Figures 3.7a and 3.7b. Single flag tiles attach to the output gate of false circuits which the flag domino (or flag tile) to attach. Circuits that have a flag tile attached will have a blocking tile attach to their right side. The output frame allows for attachment of a single test tile (Figure 3.6d). Since we do not add the test tile in this stage this attachment can occur if and only if the test tile was terminal in the previous stage (the circuit output false). The output frame then attaches to the circuit assembly. Single tiles will fill in the empty spaces left by attachment of arms to hide the information that was passed to fill the block as shown in Figure 3.4c. This process always builds one of our output frames with the only difference being whether or not a test tile has attached to the output frame. Since the test tile is in this bin if and only if the circuit output false we are able to get the output of the circuit from this final assembly.

The other assemblies that are added in this bin are the unused gates from the second stage which are terminal since they never attached to the circuit assembly. Each of these assemblies can



(a)



(b)

Figure 3.7: (a) If the circuit evaluates to true this process takes place in the third stage. First the blocking tile attaches to the test tile. Then the output frame will attach. Since the test tile was not terminal in the previous stage the output frame is empty. Single tiles then fill in the assembly to reach our output template for True. (b) If the circuit evaluates to false then in the third stage a flag tile will be able to attach to the circuit. Once this occurs the test domino will be able to cooperative attach as well. In this case the test tile was terminal in the previous stage so the output frame may attach to it and then to the circuit. Finally single tiles fill in reaching our Output template for False.

grow into a full circuit assembly with the single tiles that were added to the system. Since each tile in the circuit has at least a single side with a strength-2 glue tiles may attach to each other growing into the full circuit assembly.

Theorem 3.2.1. For any function f computed by an n-input boolean circuit with k gates, there exists a 3-stage $\mathscr{O}(n^2+k^2)$ bin, temperature-2 staged tile assembly computer that covertly computes f with an output template size of $\mathscr{O}(n^2+k^2)$.

Proof. Given any boolean circuit c, we create gate assemblies for each gate. Given the input to this circuit we create input assemblies that encode the input. In the second stage input assemblies start attaching together to form a circuit assembly. Once two inputs to a gate have attached the gate assembly computing the output of that gate is able to attach. Two gate assemblies cannot attach away from the circuit assembly. There only exist strength-1 glues on the outer edges of a gate assembly, thus a gate can only attach to another assembly with a cooperative bind at each arm. This ensures gates only attach once both inputs are present, and forces the circuit to assem-

ble in the correct order. In the second stage we add in the test tile. This test tile may only attach to a circuit assembly that has a flag tile attached. The flag tile is only present on output gates that evaluate to true so the test tile may attach if and only if the circuit evaluates to true. The test tile is terminal otherwise.

In the final bin we add in single tiles to hide the input to the circuit and the inputs to each gate. As explained above each assembly input to this bin will grow into a full circuit assembly. This full circuit assembly will grow into one of our two output frames. The output frame is a full circuit assembly with the output frame attached. The output frame contains a test tile for false and is empty for true. The terminal assembly of our staged system will have a test tile in the output frame if and only if the circuit evaluated to false which is one of our output frames. If the circuit evaluates to true the test tile will not be present.

This system uses a polynomial number of bins in the first and second stage and a single bin the final stage. The number of bins in the first and second stage are bounded by the size of our tile set since we need individual bins to store each tile so they do not combine before the final stage. The max size of a gate assembly is $\mathcal{O}(max(n,k))$ since in the worst case a gate needs to stretch across the whole circuit. The same bound applies to input bit assemblies. Therefore the size of the system (the number of tile types + the size of the mix graph) is $\mathcal{O}(n^2+k^2)$ □

3.3 Unique Assembly Verification

We now utilize covert computation to show that the open problem of Unique Assembly Verification in staged self assembly is PSPACE-complete. We start by showing UAV with 3 stages is Π_2^p-hard. We then show how to extend this construction to show that general staged UAV is PSPACE-complete. With some adjustments the same concept is used to show that when limiting the system to n stages, the problem of UAV is Π_{n-1}^p-hard.

3.3.1 3-stage UAV is Π_2^p-hard

We modify the covert computation construction to provide a reduction from ∀∃SAT. Given an instance of ∀∃SAT, we create a 3-stage temperature-2 staged system that uniquely produces a target assembly iff the given instance of ∀∃SAT is true. The reduction uses the same high-level idea as [29] and [4]. The process begins with the construction of an assembly for every input to the ∀∃SAT formula. Circuit assemblies build from these inputs and are flagged as true or false, while encoding a partial assignment through their geometry. Separate "test" assemblies are constructed that also encode a partial assignment to the same variables, which attach to true circuit assemblies with matching assignments. The systems uniquely assembles a target assembly if for all test assemblies there exists a compatible true circuit assembly for it to attach to. See Figure 3.8 for a visual overview of the created system.

Problem: ∀∃SAT

Input: An n-bit boolean formula $\phi(x_1, x_2 ... x_n)$ with the inputs divided into two sets X and Y.

Question: Does there exist an assignment to Y such that $\phi(X,Y) = 1$

First Stage

Input and Gate Assemblies. In the first stage an *input bit assembly* for both assignments to every variable $x_1, \ldots, x_n$ is built in its own bin ($2n$ bins in total). Input bit assemblies have the same structure as in the covert computation construction, except that input bit assemblies representing bits in X also have a horizontal row of tiles on the left of the frame that reflects the bit value. Figure 3.9a shows this modification to the input bit assemblies. The bit assemblies representing variable x_n no longer has additional tiles that attach to the test tile used in section 3.2. The input bit assemblies representing variable $x_{|X|+1}$ have an additional 2 tiles attached, which are used to attach to the test assembly. Gate Assemblies are built in the same way described in

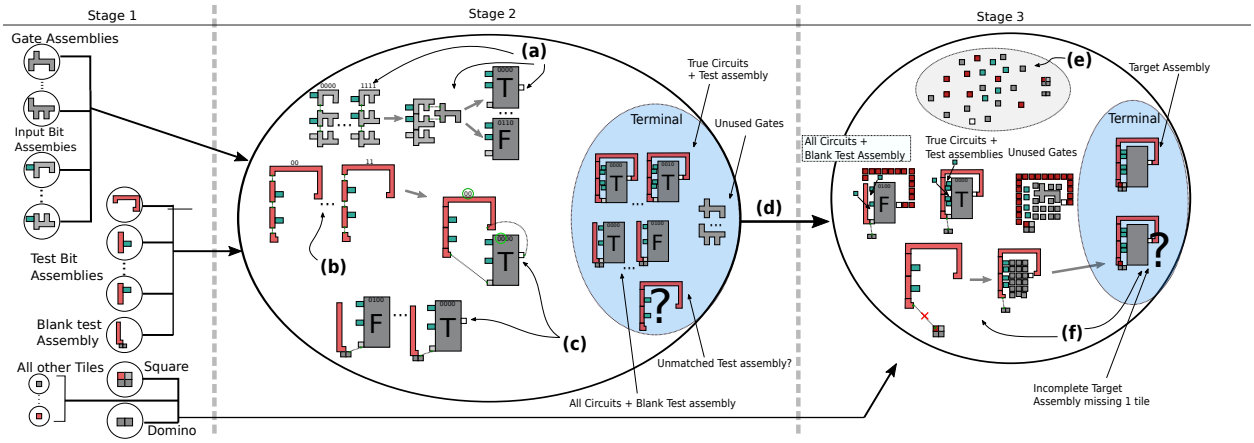


Figure 3.8: A high level overview of the staged system created from an instance of ∀∃SAT. (a) An input assembly is created for every possible input to ϕ and is evaluated using the computation technique from Section 3.2. (b) A test assembly is created for every possible input to X. (c) Test assemblies can attach to a true circuit assembly with the same assignment to X. Blank test assemblies attach to any circuit. (d) Terminal assemblies are passed to the next stage, including unmatched test assemblies if any exist. (e) In this stage we add the domino and square assemblies, as well as every other single tile of the target assembly. (f) Any unmatched test assembly will grow into an incomplete target assembly since it cannot attach to the square assembly. These incomplete target assemblies are terminal, meaning the UAV instance is false.

Section 3.2.

Test Assemblies. Similar to the input bit assemblies, two *test bit assemblies* are constructed for every variable in X. A test bit assembly is a column of connected tiles, with a horizontal row of 3 tiles extending to the right, the position of this row represents an assignment "0" or "1". An example test assembly building from separate test bit assemblies is shown in Figure 3.9c. A test assembly is composed of $|X|$ test bit assemblies. Test assemblies have additional geometry that allow them to attach to a circuit assembly.

Second Stage In the second stage the input bit assemblies will attach together nondeterministically to form 2^n unique input assemblies. The "1" and "0" input bit assembly exist for every variable, so the nondeterministic nature of the model allows for the construction of an input assembly for every possible input to the circuit. From this input assembly, computation will begin as described in the covert section. There will exist a circuit assembly for each of the 2^n possible

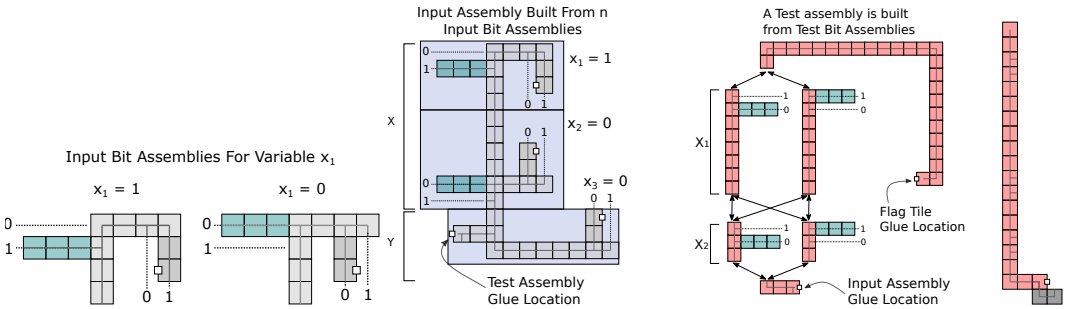
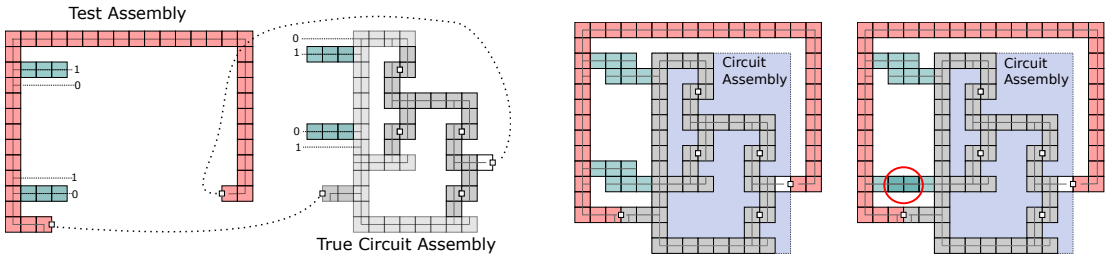


Figure 3.9: (a) Input bit assemblies for variables in X with geometry on the left reflecting the bit value. (b) An example initial circuit assembly for input $x_1 = 1, x_2 = 0, x_3 = 0$. The geometry on the left side of the assembly represents the assignment of X. (c) Separately built test bit assemblies nondeterministically attach to build one test assembly for every assignment to variables in X. (d) The blank test assembly is composed of the same base but has no protruding arms.

inputs, and each will be flagged as true or false, represented by the existence of a flag tile on the output gate. We call a circuit assembly that contains the flag tile a *true circuit assembly*.

Test Assemblies. Test bit assemblies nondeterministically combine in this stage to create a unique test assembly for every assignment to variables in X, with its assignment encoded in its geometry. A test assembly can cooperatively bind to a true circuit assembly (with the same assignment to X) by having glue strength with the output flag tile and the input assembly (Figure 3.10b). A test assembly has its assignment encoded in its geometry in a complementary fashion to that of a circuit assembly. This ensures that a test assembly is geometrically blocked from attaching to a circuit assembly that encodes a different assignment to X (Figure 3.10c). If there are no true circuit assemblies with the assignment x to X, the test assembly that represents that assignment of x will be terminal in the second stage. We refer to these as *unmatched* test assemblies.

Lemma 3.3.1. Let t_x be the test assembly representing the assignment x to the variables in X. t_x is unmatched (terminal in the second stage) if and only if for all assignments y to the variables in $Y(\phi(x,y) = 0)$.



(a) (b) (c)

Figure 3.10: (a) A test assembly (left) and a true circuit assembly that represent the same assignment to variables in X (In this construction true assemblies contain the flag tile). (b) A test assembly attaching to true circuit assembly with a matching assignment to X. (c) A test assembly that is geometrically blocked from attaching to a true circuit assembly due to having a different assignment to X. The red circled area shows the point of overlap.

Proof. True circuit assemblies are the only assemblies which have the necessary glue types to attach to t_x. The remaining question is whether a true circuit assembly exists which represents a compatible assignment. t_x can attach to any true circuit assembly with a matching assignment to X, regardless of that circuit's assignment to Y. It follows that if there exists an assignment y to Y such that $\phi(x,y) = 1$, then t_x is not terminal. The negation of which is $\forall y(\phi(x,y) = 0)$, then t_x is terminal. □

Third Stage The third stage utilizes a single bin that all assemblies are combined in. Nearly all single tiles of the target assembly are added. Four single tiles are specifically excluded, and instead two subassemblies are added in. This is done carefully to ensure the following property: every assembly except unmatched test assemblies from the second stage will grow to the target assembly. Our target assembly contains a circuit assembly attached to a test assembly with every empty spot filled in. At the point where a test assembly attaches a the circuit assembly, a domino assembly is attached completing the target assembly as seen in Figure 3.11a.

Lemma 3.3.2. Let A be the set of initial assemblies in the sole bin in the third stage. For all assemblies $a \in A$, a will grow to the target assembly iff a is not an unmatched test assembly.

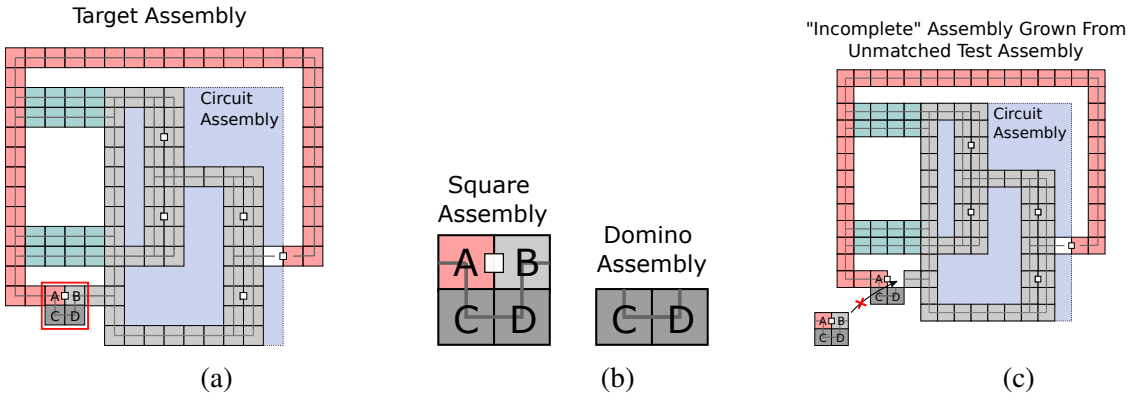


Figure 3.11: (a) An example target assembly. The area boxed in red shows where the test assembly meets the input assembly (A,B), and the adjacent domino (C,D). The four tile types A,B,C,D are not added in individually at the third stage. (b) The two additional assemblies that are added in at the third stage, composed of the same four tile types. (c) The square assembly is geometrically blocked from attaching to an assembly that grew from an unmatched test assembly, as they both contain tile A.

Proof. All individual tiles of the target assembly are added into the last stage, with the exception of four withheld tiles: the two tiles where the test assembly and input assembly meet, and the two tiles below that (tiles A,B,C,D in Fig. 3.11a). Instead of these four tiles, two assemblies are added that we refer to as the square and domino (Fig. 3.11b). These two assemblies perform the function of allowing every initial assembly besides unmatched test assemblies to grow into the target assembly. True circuit assemblies with test assemblies attached will have their empty spaces filled by single tiles, and the domino assembly will attach. Unused gates will grow to a near-complete circuit, attach to the square assembly, and then continue to grow to the target assembly. True and False Circuit Assemblies with blank test assemblies attached already contain the four withheld tiles, so will grow to the target by attaching to all necessary single tiles. Unmatched test assemblies that did not attach to a true circuit assembly can grow to a near complete target assembly, however, it will never acquire tile B (Fig. 3.11a), as it could only achieve this by attaching to the square assembly. They both contain tile A, making it geometrically blocked from doing so (Fig. 3.11c). □

Theorem 3.3.3. UAV in the Staged Assembly Model with three stages is Π_2^p-hard with $\tau = 2$.

Proof. Given an instance of $\forall\exists$SAT, the reduction provides an instance of a 3-stage temperature-2 UAV instance which is true if and only if the instance of $\forall\exists$SAT is true.

If the instance of $\forall\exists$SAT is true, then for all assignments x to X, there exists an assignment y to Y with $\phi(x,y) = 1$. By Lemma 3.3.1, this implies there will be no unmatched test assemblies. By Lemma 3.3.2, every assembly that is not an unmatched test assembly or grown from an unmatched test assembly will grow into the target assembly in the third stage. Thus, the system uniquely produces the target assembly. If the $\forall\exists$SAT instance is false, then there exists an assignment x to X, s.t. for all assignments y to Y, $\phi(x,y) = 0$. By Lemma 3.3.1, a test assembly representing assignment x would be unmatched, and by Lemma 3.3.2, unable to grow into the target assembly. Thus, this UAV instance is false. □

3.3.2 Staged UAV is PSPACE-hard

In this section, we explain at a high level how the reduction is extended to reduce from TQBF with n quantifiers over n variables to temperature-2 $\mathcal{O}(n)$-stage UAV, showing that Staged UAV is PSPACE-Hard.

Problem: *TQBF*

Input: A boolean formula ϕ with n variables $x_1, \ldots, x_n$.

Question: is it true that $\forall x_1 \exists x_2 \ldots \forall x_n (\phi(x_1, \ldots, x_n) = 1)$?

We utilize the same technique used in section 3.3.1 which reduced from $\forall\exists$SAT, a special case of *TQBF* limited to only 2 quantifiers, but adapt the technique to work with a QBF with n quantifiers $\forall x_1 \exists x_2 \ldots \forall x_n (\phi(x_1, \ldots, x_n) = 1)$. In the 3rd stage, instead of adding in single tiles to "clean up", we add in a second set of test assemblies that represent an assignment with one less variable in the next stage and are complementary in their geometry. These new test assemblies then attach to previous test assemblies that were terminal in the previous stage with matching partial assignments. This process computes an additional quantifier. We can then repeat this process

of adding in complementary sets of test assemblies for the number of quantifiers required. In the final stage, if a test assembly from the final set couldn't find a complementary test assembly to attach to, the instance of TQBF is false, and that test assembly is prevented from growing to the target assembly. This allows the truth of instance of staged UAV to correspond to the truth of the QBF. See Figure 3.12 for a depiction of the mix graph. We now show how in a certain stage the existence of a terminal test assembly relates to the truth of a statement about the boolean formula.

In total the system will have $n+1$ stages, and $n-1$ sets of test assemblies will be added (denoted $T_2, \ldots, T_n$). The set T_s will be mixed in at stage s. The first set T_2 represents an assignment to $x_1, \ldots, x_{n-1}$, and each consecutive set represents one less variable than the set before it, i.e., a test assembly $t_s \in T_s$ represents a partial assignment to $x_1, \ldots, x_{n-s-1}$. The sets alternate between type L and R, which correlates to the direction the arms face (Compare T_3 and T_4 in Figure 3.12). We build all these sets of test assemblies using the same method in the first stage, and pass them along through "helper" bins until they are needed.

Five Stages We build off the reduction in Section 3.3.1. Assume we are given an instance of TQBF P over n variables, we will create an instance of $n+1$-stage UAV P' such that P' is true $\iff$ P is true. We focus on a small example where $n = 4$, i.e., $P_4 = \forall x_1 \exists x_2 \forall x_3 \exists x_4 (\phi)$ and create a 5-stage instance of staged UAV P'_4. Afterwards we explain prove the results hold in general. The example instance of Staged UAV will have five stages. Given the TQBF instance P_4, an instance of $\forall\exists\text{SAT} = \forall x_1, x_2, x_3 \exists x_4 (\phi)$ is created. With the instance of $\forall\exists$SAT this we use the reduction in Section 3.3.1 to create an instance of 3-stage UAV. We keep the first two stages the reduction creates, and ignore the third.

By Lemma 3.3.1, a test assembly representing the partial assignment $x_1, \ldots, x_3$ is terminal if and only if $\forall x_4 (\phi(x_1, x_2, x_3, x_4) = 0)$. We refer to this set of test assemblies as T_2.

Additional Test Assemblies. Two more sets of test assemblies will be added, one in stage 3 and one in stage 4. We will call these sets T_3 and T_4 respectively. These test assemblies

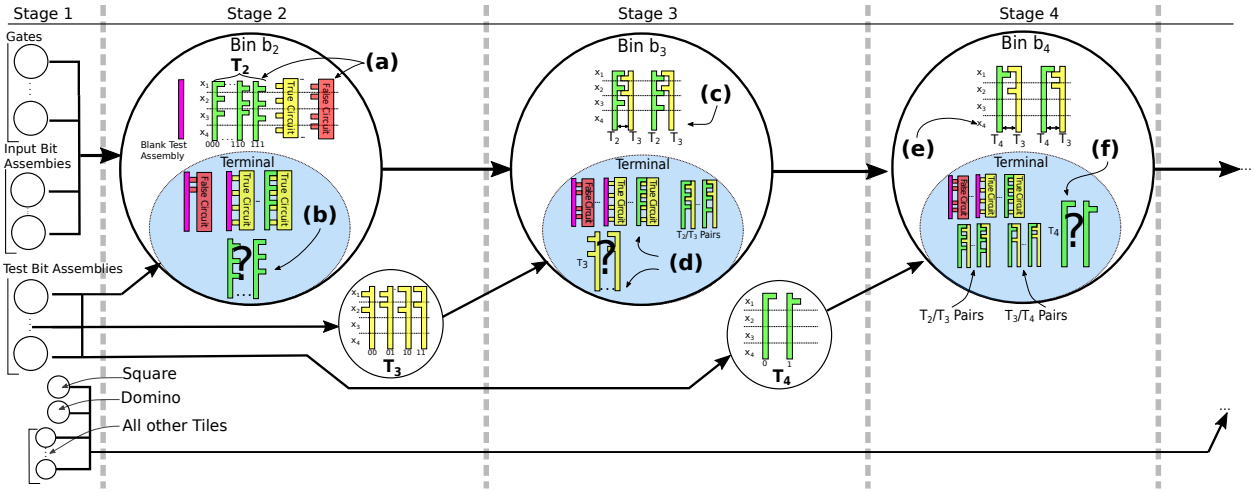


Figure 3.12: An example mix graph for an instance of TQBF with 4 variables. (a) Test bit assemblies combine into T_2 test assemblies. Circuit assemblies evaluate every input. (b) T_2 test assemblies attach to compatible (matching partial assignment) true circuits. Any unmatched T_2 assemblies are passed to the next stage. (c) T_3 test assemblies are added in and attach to compatible T_2 test assemblies. (d) Any unmatched T_3 assemblies are passed to the next stage. (e) T_4 test assemblies are added and attach to compatible T_3 test assemblies. (f) The existence of an unmatched T_4 assembly directly corresponds to the truth of the TQBF instance.

will be a subassembly of our *total test assembly* (Figure 3.13d). We say T_3 are type-R test assemblies, meaning their base is on the left and their variable arms protrude right (Figure 3.13a). Similarly, T_4 are type-L test assemblies (Figure 3.13b). Type L and R test assemblies encode their assignment in a complementary fashion. This allows them to attach to each other only if they encode the same partial assignment (Figure 3.13c). The "arms" for these test assemblies will be length 4, allowing them to attach to each other without cooperative binding. The set T_2 contains eight test assemblies, one for each assignment to x_1, x_2, x_3. The set T_3 contains four test assemblies, each representing an assignment to x_1, x_2. The test assembly set T_4 will have two test assemblies, each representing an assignment to x_1. This requires adding additional bins to the first stage to create additional test bit assemblies.

Mix graph. Figure 3.12 depicts the mix graph structure. In stages 3 and 4 there is one bin which contains the circuit assemblies (bins b_3 and b_4). In the third stage we mix the set of test assemblies T_3 into bin b_3. These will "search" for unmatched T_2 test assemblies to attach to.

The set $T_3' \subseteq T_3$ for which all $t_3' \in T_3'$ are terminal in bin b_3 gives us information about the partial assignments that the t_3' assemblies represent.

W.l.o.g., consider the test assembly $t_{1,0} \in T_3$ representing partial assignment $x_1 = 0, x_2 = 1$. $t_{1,0}$ will not be terminal in bin b_3 if there exists some unmatched $t_2 \in T_2$ it can attach to, i.e., a test assembly t_2 that represents the same partial assignment $x_1 = 0, x_2 = 1$. We know there are two of these, one representing assignment $x_1 = 0, x_2 = 1, x_3 = 0$, and one representing assignment $x_1 = 0, x_2 = 1, x_3 = 1$.

By Lemma 3.3.1, the condition for a test assembly $t_{0,1,x_3} \in T_2$ representing assignment $x_1 = 0, x_2 = 1, x_3$ to be unmatched is $\forall x_4(\phi(0,1,x_3,x_4) = 0)$. If this is false for all assignments to x_3 ($x_3 = 0$ and $x_3 = 1$), $t_{1,0}$ will have nothing to attach to. Therefore $t_{1,0} \in T_3$ is terminal if for $\forall x_3 \exists x_4(\phi(1,0,x_3,x_4) = 1)$. More generally a test assembly $t_3 \in T_3$ representing assignment x_1, x_2 is terminal in bin b_3 if $\forall x_3 \exists x_4(\phi(x_1,x_2,x_3,x_4) = 1)$.

Let $TERM(A,b)$ be true iff assembly A is terminal in bin b. The set of test assemblies T_4 is then mixed into bin b_4. Utilizing the same logic, a test assembly $t_4 \in T_4$ representing assignment x_1 is terminal in bin b_4 if and only if $\forall x_2 \exists x_3 \forall x_4(\phi(x_1,x_2,x_3,x_4) = 0)$. We can then say $\exists t_4 \in T_4(TERM(t_4,b_4)) \iff \exists x_1 \forall x_2 \exists x_3 \forall x_4(\phi(x_1,x_2,x_3,x_4) = 0)$. Therefore the existence of an unmatched t_4 test assembly directly corresponds to the truth of the instance of TQBF P_4.

Target Assembly. Stage 5 has one bin b_5. It has been shown that the existence of an unmatched test assembly $t_4 \in T_4$ corresponds to the truth of the QBF. It suffices that in bin b_5, if all other initial assemblies can grow to the target assembly while unmatched T_4 test assemblies can not, then the created instance of UAV corresponds to the truth of the QBF. Conditionally growing assemblies to the target assembly is done the same way as in section 3.3.1 (See Figure 3.13d), flooding all tiles except a select few and adding in a square and domino assembly built from those tiles instead. This allows the instance of staged UAV to directly correspond to the truth of the TQBF instance P_4.

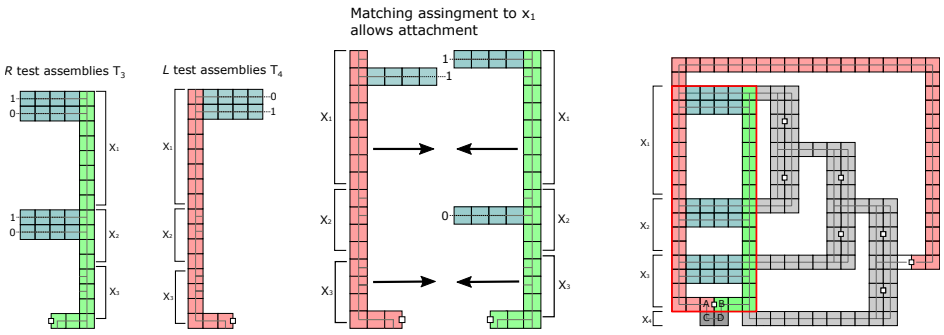


Figure 3.13: (a) The set T_3 of test assemblies will contain all R test assemblies created by choosing one arm for each variable ($|T_3| = 4$). (b) The set T_4 will contain 2 test assemblies, one encoding $x_1 = 0$ and one encoding $x_1 = 1$. (c) An R and L test assembly can attach provided they encode the same partial assignment. (d) An example target assembly for an instance of TQBF where ϕ is over 4 variables. The *total* test assembly is outlined red. The four withheld tiles A, B, C, D are not *not* flooded individually in the last stage.

General To scale this reduction to a QBF over n variables, we will utilize more stages and sets of test assemblies. Given the TQBF instance P, we will ensure that it is of the form $\forall x_1, \ldots, \exists, x_n(\phi)$. In the case where it is not of this form we add "dummy" variables and quantifiers which don't effect the truth of the statement. This can be done without increasing n by more than 2.

Test Assemblies. In total n sets of test assemblies will be added. T_2 represents an assignment to $x_1, \ldots, x_{n-1}$, and each consecutive set represents one less variable than the set before it, i.e., a test assembly $t_s \in T_s$ represents a partial assignment to $x_1, \ldots, x_{n-s-1}$. T_3 is composed of R test assemblies, the following sets alternate between type L and R. We build all these test assemblies using the same method, making test bit assemblies for each variable and allowing them to nondeterministically combine.

Mix graph. In general the mix graph will have $n+1$ stages. At every stage $s \geq 2$, the bin that contains the circuit assemblies (see Figure 3.12) in that stage is referred to as b_s. The set of test assemblies T_s is mixed into bin b_s. We say a test assembly $t_s \in T_s$ is *unmatched* if t_s is terminal in bin b_s (If t_s is not in the final set T_n, we know it will not be terminal in later stages).

We design our system in such a way to achieve a generalization of Lemma 3.3.1. In general, for a test assembly $t_s \in T_s$ which represents a certain partial assignment $x_1, \ldots, x_a$, there is statement S that quantifies over the rest of the variables (e.g. $S = \forall x_{a+1}, \ldots, \exists x_n(\phi(x_1, \ldots, x_n) = 1)$) such that $S \iff t_s$ is terminal in bin b_s.

Lemma 3.3.4. Let $TERM(A, b) \iff$ (Assembly A is terminal in bin b). Let a be the number of variables the test assemblies in T_s represent ($a = n - s + 1$). Let $t_s(x_1, \ldots, x_a)$ be the test assembly $t_s \in T_S$ that represents partial assignment $x_1, \ldots, x_a$. In the staged system S_P created from an instance of TQBF P over n variables: $\forall s \in \{1, \ldots, n\}(TERM(t_s(x_1, \ldots, x_a), b_s) \iff \forall x_{a+1} \exists x_{a+2}, \ldots, Qx_n(\phi(x_1, \ldots, x_n) = y))$. If s is even, $y = 0$ and $Q = \forall$, and $y = 1$, $Q = \exists$ otherwise.

Proof. We prove this by induction on s. Base case: Lemma 3.3.1

Let Q' be the opposite quantifier of Q. Assume the statement holds for case $s-1$, we will show this implies it holds for case s. Since we are assuming the case $s-1$ the following statement holds:

$$TERM(t_{s-1}(x_1, \ldots, x_{a+1}), b_{s-1}) \iff \forall x_{a+2}, \ldots, Q' x_n(\phi(x_1, \ldots, x_n) = \neg y)$$

In the next stage, test assemblies in T_s represent one less variable, i.e., a partial assignment $x_1, \ldots, x_a$ and search for a test assembly in T_{s-1} with a matching partial assignment to attach to. Consider test assembly $t_s(x_1, \ldots, x_a)$ in bin b_s. $t_s(x_1, \ldots, x_a)$ is not terminal in bin b_s if and only if either $t_{s-1}(x_1, \ldots, x_a, 0)$ or $t_{s-1}(x_1, \ldots, x_a, 1)$ were not terminal in bin b_{s-1} (Otherwise geometric blocking would prevent attachment). If follows that $t_s(x_1, \ldots, x_a)$ is terminal in bin b_s if for all $x_{a+1}(t_{s-1}(x_1, \ldots, x_{a+1})$ is *not* terminal in bin $b_{s-1}))$, therefore:

$$TERM(t_s(x_1, \ldots, x_a), b_s) \iff \forall x_{a+1}(\neg TERM(t_{s-1}(x_1, \ldots, x_a, x_{a+1}), b_{s-1}))$$

Since we assumed $TERM(t_{s-1}(x_1,\ldots,x_{a+1}),b_{s-1}) \iff \forall x_{a+2},\ldots,Q'x_n(\phi(x_1,\ldots,x_n) = \neg y)$ we substitute the $TERM$ function for the corresponding statement, we then simplify to show the Lemma's statement holds for case s:

$$TERM(t_s(x_1,\ldots,x_a),b_s) \iff \forall x_{a+1}(\neg(\forall x_{a+2},\ldots,Q'x_n(\phi(x_1,\ldots,x_n) = \neg y)))$$

$$TERM(t_s(x_1,\ldots,x_a),b_s) \iff \forall x_{a+1}(\exists x_{a+2},\ldots,Qx_n(\phi(x_1,\ldots,x_n) = y))$$

$$TERM(t_s(x_1,\ldots,x_a),b_s) \iff \forall x_{a+1}\exists x_{a+2},\ldots,Qx_n(\phi(x_1,\ldots,x_n) = y)$$

□

We now show a generalization of Lemma 3.3.2. Since it was shown that the existence of an unmatched test assembly t_n in bin b_n corresponds to the truth of the instance of TQBF, we now show how in bin $b_n + 1$, every assembly that is not an unmatched t_n will grow to the target assembly, while t_n will not.

Lemma 3.3.5. In the staged system S_P created from an instance of TQBF P over n variables, in bin b_{n+1} in stage $n+1$, let A be the set of initial assemblies in b_{n+1}. For all $a \in A$, a will grow to the target assembly if and only if a is not an unmatched test assembly $t_n \in T_n$.

Proof. This is accomplished in a similar way to the previous section. In stage $n+1$ all individual tiles of the target assembly are added except for four tiles we refer to as the *withheld* tiles (tiles A,B,C,D Figure 3.14). Two assemblies are added, which we refer to as the Square and Domino. The Square assembly is composed of all four withheld tiles, the Domino is composed of tiles C and D.

There are a number of initial assemblies passed in bin b_{m+1} to account for, and each of these are subassemblies of the target assembly.

1. False circuit assemblies with a blank test assembly attached (Figure 3.14a).

2. True circuit assemblies with a blank test assembly attached (Figure 3.14b).

3. True circuit assemblies with a $t_2 \in T_2$ test assembly attached (Figure 3.14c).

4. t_a and t_{a-1} test assembly pairs that attached (Figure 3.14d).

5. Unused Gates (Figure 3.14e).

6. Single Tiles, the Square assembly, and the Domino assembly.

We will explain how each of these categories of assemblies grows to the target assembly. Note that any subassembly of the target assembly that contains the four withheld tiles trivially grows to the target assembly, as every tile it is missing now exists alone, and will eventually attach to it. Therefore if an assembly can acquire the four withheld tiles it is guaranteed to grow to the target. Items 1,2, and the Square assembly already contain them, so they will grow to the target assembly.

If an assembly already contain tiles A and B, but not C and D, then the domino assembly will attach to it. It now contains all four withheld tiles. This accounts for items 3 and 4. If it contains none of the withheld tiles, then eventually the square assembly will attach to it, it then contains all four withheld tiles. This accounts for item 5 and all single tiles. Lastly, he domino assembly will attach to all assemblies in category 3 or 4, at least one of which is guaranteed to exist, and will then contain the four withheld tiles.

We now look at how an unmatched test assembly $t_n \in T_n$ will grow (shown in Figure 3.14f). Note that an unmatched t_m test assembly will exclusively contain tile A, and needs to acquire tile B to grow to the target assembly. No other unmatched R test assemblies exist in this bin for t_m to attach to. It can not attach to the square assembly, since both t_m and the square assembly both contain tile A they are geometrically blocked from attaching. Therefore t_n will never grow to the target assembly. □

Theorem 3.3.6. Unique Assembly Verification in the Staged Assembly Model is PSPACE-complete with $\tau = 2$.

Proof. Given an instance of TQBF P over n variables/quantifiers, the reduction provides an instance of $n+1$-stage $\tau = 2$ UAV that is true if and only if P is true. If P is true, then in stage $n+1$, every producible assembly grows into the target assembly. Since n is always even, by Lemma 3.3.4, for a bin b_n in stage n, an assembly $t_n \in T_n$ representing an assignment x_1 is terminal in bin b_n if $\forall x_2 \exists x_3, \ldots, \forall X_n(\phi(x_1, \ldots, x_n) = 0)$. If P is true, then the statement $\forall x_1 \exists x_2 \forall x_3, \ldots, \forall X_n(\phi(x_1, \ldots, x_n) = 1)$ is true, and therefore no unmatched $t_n \in T_n$ will be passed into b_{n+1}. By Lemma 3.3.5, every initial assembly in b_{n+1} that is not some $t_n \in T_n$ grows into the target assembly. Therefore, the target assembly is uniquely assembled if the instance of TQBF is true. If P is false, then there exists an assignment to x_1 such that $\forall x_2 \exists x_3, \ldots, \forall X_n(\phi(x_1, \ldots, x_n) = 0)$. By Lemma 3.3.4, some test assembly $t_n \in T_n$ will be terminal and passed into bin b_{n+1}. By Lemma 3.3.5, any $t_n \in T_n$ will not grow into the target assembly. Thus, the instance of staged UAV is false. □

3.3.3 n-Stage Hardness

We now show how the reduction can be used to show hardness for n-stage UAV. We reduce from the boolean satisfiability problem for Π_n^p, which is a quantified boolean formula with n quantifiers (starting with universal) and $n-1$ alternations. We show an instance of Π_n^p-SAT can be reduced to $n+1$-stage $\tau = 3$ UAV.

Problem: $\Pi_n^p - SAT$

Input: A boolean formula ϕ with variables partitioned into n sets $X_1, \ldots, X_n$.

Question: Is it true that $\forall X_1 \exists X_2 \ldots Q_n X_n(\phi(X_1, \ldots, X_n))$?

Theorem 3.3.7. For all $n > 1$, UAV in the Staged Assembly Model with n stages is Π_{n-1}^p-hard with $\tau = 2$.

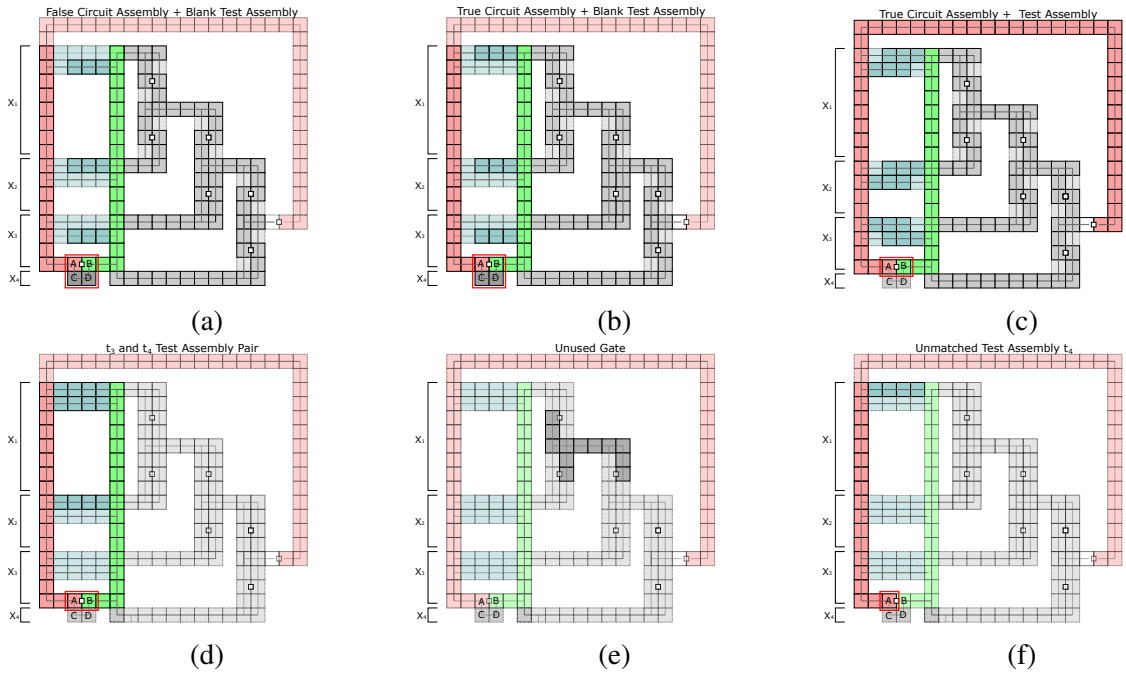


Figure 3.14: Example initial assemblies in bin b_5 in a system created from a QBF over 4 variables. The transparent section shows the tiles it is missing from the target assembly. (a) A False Circuit assembly with a blank test assembly attached already contains the four withheld tiles. (b) A True Circuit assembly with a blank test assembly attached already contains the four withheld tiles. (c) A True Circuit assembly with a $t_2 \in T_2$ test assembly attached already contains tiles A and B, the domino assembly will attach. (d) A t_4 and t_3 pair that attached already contains tiles A and B, the domino will attach. An unused gate contains none of the four withheld tiles, so the square assembly can attach. (e) An unmatched T_4 assembly already contains tile A, but can never acquire tile B.

Proof. Assume we are given an instance of Π_n^p-SAT, $P = X_1, \ldots, X_n, \phi$. Each X_i is a set of variables. The reduction creates an instance of $n+1$-stage $\tau = 3$ UAV that is true if and only if P is true. First note that Lemma 3.3.4 can extend to the case where each set of test assemblies represents one less set of variables, rather than one less variable.

If n is even, the system behaves as previously described. By Lemma 3.3.4, in stage n the set T_n is added into b_n. Any $t_n \in T_n$ representing an assignment to the variables in X_1 is terminal if and only if $\forall X_2 \exists X_3 \ldots \forall X_n (\phi(X_1, \ldots, X_n) = 0)$. In b_{n+1} all assemblies besides any t_n grow to the target assembly.

If n is odd, then by Lemma 3.3.4 any $t_n \in T_n$ representing an assignment to X_1 is terminal if $\forall X_2 \exists X_3 \ldots \exists X_n(\phi(X_1, \ldots, X_n) = 1)$. However, since we modified the output assemblies to contain the flag tile if they represent a *false* output, they are now terminal if the statement is true for the negation of ϕ. Therefore any t_n representing X_1 is terminal if and only if $\forall X_2 \exists X_3 \ldots \exists X_n(\phi(X_1, \ldots, X_n) = 0)$. In bin b_{n+1} all assemblies besides any t_n grow to the target assembly in the same way. □

CHAPTER IV

THE K-HANDED ASSEMBLY MODEL

In this section we investigate the complexity of fundamental problems related to multiple handed self-assembly. Our focus here is a generalization of the 2HAM, called the k-HAM, which allows groups of up to k assemblies (one in each of up to k hands) to combine to create new stable assemblies. See [25] for a general survey of tile self-assembly, and [33] for a survey of intrinsic simulation in tile self-assembly, and [12, 35] for an overview of algorithmic self-assembly and recent experimental implementations in DNA.

The most fundamental version of self-assembly may be the basic 2-handed model as the statistical likelihood of multiple pieces combining within a short enough time window to stabilize is small in many experimental models of self-assembly. However, such productions do happen in practice and are often referred to as undesirable *spurious nucleation reactions* [23]. Moreover, at certain scales it is useful to consider the landscape of *stable* configurations, especially *local minimum energy* configurations, even when reaching such configurations would require moving more than two things into place simultaneously. Multiple handed self-assembly provides a simple framework for exploring self-assembly phenomena that could have impacts in these scenarios. In particular, multiple handed self-assembly may prove to be a useful tool for designing robustness in self-assembly systems, e.g., designing a system that is guaranteed to work even if the number of hands is increased up to some value k. Applications of such robustness theory would be directly aided by understanding fundamental complexities of self-assembly with multiple hands. There are also classes of shapes that may not be built efficiently without multiple hands such as

certain fractals that rely on multiple shapes coming together simultaneously [9], and there are shapes that cannot be built as efficiently at lower temperatures. Thus, given different experimental constraints, a reliance on these interactions with fewer tiles at a lower temperature may be preferable.

4.1 Preliminaries

Tiles. A *tile* is a non-rotatable unit square with each edge labeled with a *glue* from a set Σ. Each pair of glues $g_1, g_2 \in \Sigma$ has a non-negative integer *strength* $\text{str}(g_1, g_2)$.

Configurations, bond graphs, and stability. A *configuration* is a partial function $A : \mathbb{Z}^2 \to T$ for some set of tiles T, i.e. an arrangement of tiles on a square grid. For a given configuration A, define the *bond graph* G_A to be the weighted grid graph in which each element of $\text{dom}(A)$ is a vertex, and the weight of the edge between a pair of tiles is equal to the strength of the coincident glue pair. A configuration is said to be τ-*stable* for positive integer τ if every edge cut of G_A has strength at least τ, and is τ-*unstable* otherwise.

Assemblies. For a configuration A and vector $\vec{u} = \langle u_x, u_y \rangle$ with $u_x, u_y \in \mathbb{Z}^2$, $A + \vec{u}$ denotes the configuration $A \circ f$, where $f(x,y) = (x + u_x, y + u_y)$. For two configurations A and B, B is a *translation* of A, written $B \simeq A$, provided that $B = A + \vec{u}$ for some vector $\vec{u}$. For a configuration A, the *assembly* of A is the set $\tilde{A} = \{B : B \simeq A\}$. An assembly $\tilde{A}$ is a *subassembly* of an assembly $\tilde{B}$, denoted $\tilde{A} \sqsubseteq \tilde{B}$, provided that there exists an $A \in \tilde{A}$ and $B \in \tilde{B}$ such that $A \subseteq B$. An assembly is τ-*stable* provided the configurations it contains are τ-stable. Assemblies $\tilde{A}$ and $\tilde{B}$ are τ-*combinable* into an assembly $\tilde{C}$ provided there exist $A \in \tilde{A}$, $B \in \tilde{B}$, and $C \in \tilde{C}$ such that $A \cup B = C$, $A \cap B = \varnothing$, and $\tilde{C}$ is τ-stable. Let the shape of an assembly A be shape taken from the set of points in $dom(A)$.

***k*-handed assembly.** The k-handed assembly model is a generalization of two-handed assembly model. A k-handed assembly system $\Gamma' = (T, k, \tau)$ is an ordered tuple where T is the *tile set*, k is the number of hands that can be used to produce an assembly, and τ is a positive integer

parameter called the *temperature*. For a system Γ', the set of *producible* assemblies $P'_{\Gamma'}$ is defined recursively as follows:

1. $S \subseteq P'_{\Gamma'}$.
2. For $2 \leq k' \leq k$, if $\{A_1, A_2, \ldots, A_{k'}\} \subset P'_{\Gamma'}$ are τ-combinable into C, then $C \in P'_{\Gamma'}$.

A producible assembly is *terminal* provided it is not τ-combinable with any other producible assembly, and the set of all terminal assemblies of a system Γ is denoted P_Γ. Intuitively, P'_Γ represents the set of all possible assemblies that can self-assemble from the initial set T, whereas P_Γ represents only the set of supertiles that cannot grow any further. The assemblies in P_Γ are *uniquely produced* iff for each $x \in P'_\Gamma$ there exists a corresponding $y \in P_\Gamma$ such that $x \sqsubseteq y$. Thus unique production implies that every producible assembly can be repeatedly combined with others to form an assembly in P_Γ.

Unique Production of Shapes and Assemblies. A system Γ uniquely assembles an assembly A if the system uniquely produces set P'_Γ that contains only the assembly A and no other assemblies. In other words all producible assemblies can be combined to eventually form A. We say a system uniquely assembles a shape $\mathscr{S}$ if the system uniquely produces set P'_Γ and for all $B \in P'_\Gamma$, B has the shape $\mathscr{S}$.

Definition 4.1.1 (Producibility Problem)**.** Given a k-HAM system $\Gamma = (T, k, \tau)$ and an assembly A, is A a producible assembly of Γ?

Definition 4.1.2 (Unique Assembly Verification Problem (UAV))**.** Given a k-HAM system $\Gamma = (T, k, \tau)$ and an assembly A, is A uniquely produced by Γ?

4.2 k-Hand Producibility

Here we show that verifying producibility of an assembly is solvable in the k-handed model in polynomial time. The proof is a modification of the proof of 2-handed producibility in [13] generalized to k-handed assembly. A partition of a configuration C is a set of unique

configurations $\mathscr{C} = \{C_1, C_2, \ldots, C_n\}$ such that $\bigcup_{i=1}^{h} C_i = A$ and for all $i \neq j$, $C_i \bigcap C_j = \emptyset$. With regards to a partition of an assembly A, we mean the partition of an arbitrary configuration $C \in A$.

Definition 4.2.1 (k-handed Assembly Tree)**.** An k-Handed Assembly Tree for a configuration C is a tree Υ where the root represents C, every other node represents a configuration $c \subseteq C$, every parent node has at most k children, and every parent node p has the characteristic that it's children are τ-combinable in an k-handed assembly step into p.

Lemma 4.2.2. For any k-handed system $\Gamma = (T, h, \tau)$ and partition C of assembly $A \in P'_\Gamma$, if $\forall a \in C, a \in P'_\Gamma$ then there exists a subset s of C such that $2 \leq |s| \leq h$ and the elements of s are τ-combinable into an assembly $B \sqsubseteq A$.

Proof. Since $A \in P'_\Gamma$, there must exist an k-handed assembly tree Υ. We utilize a method from [13] to mark nodes in Υ to find a valid candidate assembly B. The previous proof used a more generalized version of an assembly tree.

Label each leaf $\{x\}$ in Υ with the unique element $C_i \in \mathscr{C}$ where $x \in C_i$. Then iteratively, if all siblings have the same label, label the parent C_i as well. This preserves the partition labels for each parent as long as it is a proper subset of the partition.

Doing a breadth first search from the root looking at only unlabeled vertices, we reach one of the 3 following cases,

1. The children have $2 \leq b \leq h$ distinct labels and there are b children.
2. The children have $2 \leq b \leq k-1$ distinct labels and there are $3 \leq c \leq h$ children and $c > b$.
3. There are labeled and unlabeled children or all unlabeled children. We ignore these nodes since there must exist nodes from either Case 1 or 2 if we follow the unlabeled children since all leaves are labeled.

Case 1. This case shows that there must exist b partitions that neighbor each other in the tree and can be brought together with b hands. There is a subtle subcase that for the parent

assembly node p, some number of the children could be combined with fewer hands. However, a modification to the build path in this way does not change p since it could be built from the single b-handed operation or through multiple joins of less than b hands since each operation is joining subsets of different partitions and p can still be formed as a stable assembly.

Case 2. Even though $3 \leq c \leq k$ hands are needed, some of the nodes have the same label. Thus, the number of distinct partition subsets is $2 \leq b \leq k-1$. Similar to Case 1, some of the children could be combined without assembling all b children at once. Any stable combination of children represents another valid k-handed assembly sub-tree for the parent node.

Let s' be the set of configurations represented by the children of the found node. For each element in s' replace it with the element of C it was labeled by (only once for each label) to form the set s. This replacement preserves the ability for all the assemblies in s to be combinable. Since we know each element of s are producible, the assembly B is producible where the elements of s are τ-combinable into B. □

Algorithm 1 The naïve method of verifying whether an assembly is producible in an k-handed system.

```
1:  procedure PROD(k-handed assembly system Γ = (T, h, τ), and an assembly A)
2:      𝒞 ← {{v} | v ∈ dom(A)}
3:      while |C| > 1 do
4:          if ∃ 2 ≤ b ≤ h subassemblies in 𝒞 (denoted C_i ∈ 𝒞 with 1 ≤ i ≤ b) s. t. ∪_{1≤i≤b} C_i is stable then
5:              𝒞 ← 𝒞 \ {C_1, ..., C_b} ∪ {∪_{1≤i≤b} C_i}
6:          else
7:              reject
8:          end if
9:      end while
10:     accept
11: end procedure
```

Theorem 4.2.3. The producibility problem for a system $\Gamma = (T, h, \tau)$ and assembly A is solvable in $\mathcal{O}(|A|^2 h \log |A|)$ time.

Proof. Algorithm 1 gives the naïve method for building the shape by combining tiles from the shape whenever possible. We know from Lemma 4.2.2 that if the target assembly A is producible, there must exist up to k subassemblies that may be combined at each step.

The runtime is affected by the time required to find cycles in planar graphs. In order for assemblies to be combined they must be adjacent. Any assembly step that requires more than 2 hands must form a loop. Thus, the bottleneck is checking for a cycle of size k in a planar graph, which varies based on k. We must also check for cycles up to size k, so we might require k calls to this subroutine. Let T be the time to find a cycle, then the runtime of Algorithm 1 is $\mathcal{O}(Th|A|)$.

Arbitrary fixed length cycles in planar graphs with n nodes can be found in $\mathcal{O}(n \log n)$ time (with expected $\mathcal{O}(n)$ time), and for any $h \leq 6$, the complexity is $\mathcal{O}(n)$ [22, 3]. Thus, for an unknown k, the runtime of Algorithm 1 is $\mathcal{O}(|A|^2 h \log |A|)$ as the size of the graph is the size of the assembly.

We note that there is a $\mathcal{O}(n)$ algorithm to find any fixed subgraph H in a planar graph, but it requires an extremely large constant that is generally considered impractical even for small n [15]. Also in the special case of 2-handed assembly the runtime of the algorithm shown in [13] runs in time $\mathcal{O}(n \log^2 n)$. □

4.3 k-Hand Unique Assembly Verification

In this section, we investigate the complexity of the problem of verifying an assembly is uniquely assembled by a given k-HAM system. We consider two different methods of encoding the number of hands in the system. We show that the problem is coNEXP-complete and coNP-complete when the number of hands is encoded in binary and unary, respectively. The problems are listed below. We first show membership, then prove hardness with a reduction from K-A_{NTM}.

Problem: K-HAM-UAV

Input: An k-HAM system $\Gamma = (T, h, \tau)$ where the integer k is encoded in binary, and an assembly A.

Question: Does Γ unique produce the assembly A?

Problem: U-K-HAM-UAV

Input: An k-HAM system $\Gamma = (T, h, \tau)$, where the integer k is encoded in unary, and an assembly A.

Question: **Output:** Does Γ unique produce the assembly A?

4.3.1 Membership

The UAV problem is in the class coNEXP if the number of hands is specified in binary, and in coNP if is specified in unary. For an instance of UAV $(\Gamma = (T, h, \tau), A)$, the instance is true if and only if the following 3 conditions are true: 1) The target assembly A is producible, 2) there does not exist a terminal assembly $C \sqsubset A$, and 3) there does not exist a producible assembly $B \not\sqsubseteq A$.

Lemma 4.3.1. K-HAM-UAV $\in$ coNEXP.

Proof. We provide a coNEXP algorithm for K-HAM-UAV that checks the above three conditions. By Theorem 4.2.3, condition 1 can be decided in polynomial time. Utilizing Lemma 4.2.2, we show that if condition 1 is true then condition 2 is true. For some assembly $C \sqsubset A$, consider any partition of the assembly A where C is an element, and by repeatedly applying Lemma 4.2.2, continue joining elements of this partition until A is built. To do this, C must, at some point, attach to another assembly. Therefore, C is not terminal.

The remaining task to decide the instance of UAV is to verify that there does not exist a producible assembly $B \not\sqsubseteq A$. A coNEXP machine can do this by nondeterministically attempting to build an assembly up to size $h|A|$. If any branch builds some assembly $B \not\sqsubseteq A$, then the branch (and machine) will reject. It suffices to check only up to this size, as an assembly of size $> h|A|$ must

have been built from at least one assembly of size $> |A|$. That assembly itself is not a subassembly of A, and therefore if it exists, then a different branch of the computation will build it and reject. Since k is encoded in binary it takes exponential time to build an assembly of size $h|A|$. □

Corollary 4.3.2. U-K-HAM-UAV $\in$ coNP.

Proof. The proof of this is the same algorithm provided in Lemma 4.3.1. Since the integer k is encoded in unary, nondeterministically building an assembly of up to size $h|A|$ takes polynomial time. □

4.3.2 Hardness: Reduction from K-A$_{NTM}$

To show coNEXP-hardness we reduce from the canonical complete problem for NEXP, K-A$_{NTM}$, which is the problem of deciding if there exists a computation path of length $\leq k$ where a given nondeterministic Turing machine M accepts when run on the empty tape. We first overview the construction, and then prove correctness. The formal problem definitions follow.

Problem: K-A$_{NTM}$

Input: A nondeterministic Turing machine M, and an integer k encoded in binary.

Question: Does there exist a computation path of M accepting within k steps when run on an empty tape?

Problem: U-K-A$_{NTM}$

Input: A nondeterministic Turing machine M, and an integer k encoded in unary.

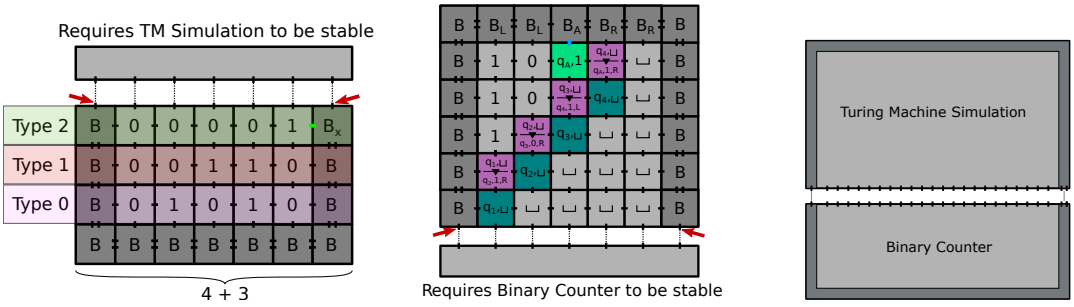
Question: Does there exist a computation path of M that accepts within k steps when run on an empty tape?

Given an instance of K-A$_{NTM}$. We will refer to the integer from the instance of K-A$_{NTM}$ as k' and the number of hands in our created system as k. We create an instance of K-HAM-UAV such that the system is temperature-4, and the number of hands k is set to $(2^{\lceil \log_2(k') \rceil} + 3) \cdot (k' + \lceil \log_2(k') \rceil + 4)$. The system always builds a specific target assembly. If and only if the answer to

K-A_{NTM} is yes, the system also produces a *computation assembly*, an assembly that represents an accepting computation path of M that is less than k' steps. The computation assembly is *not* the target assembly, making the answer to the UAV instance 'no.' We will walk through an example, reducing from an instance $(M, k' = 4)$ and creating a temperature-4 system where the number of hands is set to $(4+3)\cdot(4+2+4) = 70$.

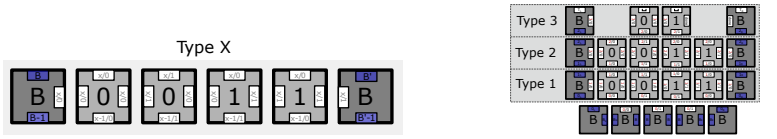
We first explain the case when the instance of K-A_{NTM} is true, and therefore the created instance of K-HAM-UAV is false. In this case, a computation assembly is built. A computation assembly is composed of a binary counter section and a Turing machine simulation section (Figure 4.1c). Since there exists an accepting computation path of less than or equal to 4 steps, then in one production step, utilizing the large number of hands in the system, ≤ 70 tiles will come together forming a tableau that represents a simulation of the computation path, as well as a binary counter enforcing the simulation to maintain a certain tape width.

Binary Counter. We utilize known techniques, such as in [26], for implementing a self-assembling binary counter where the construction has a size-$\mathcal{O}(\log_2 k')$ tile set (shown in Figures 4.2a and 4.2b) and bounds the counter such that it stops once it reaches $2^{\lceil \log_2(k') \rceil}$. For our example instance $(M, k' = 4)$, the assembled binary counter is shown in Figure 4.1a. This assembly is one of two parts of the computation assembly, and is not stable by itself at temperature 4. The binary counter counts from left to right, starting at 0 and ending at $2^{\lceil \log_2(k') \rceil}$. The bottom row of light gray tiles represents the least significant bit, while the top row represents the most significant bit. Each row uses a distinct set of tiles, preventing unbounded growth. The majority of the tiles (light gray) have a strength-1 glue on each side. Thus, these tiles are adjacent to a tile on every side in order to be stable in the assembly. The remaining border tiles (dark gray) are the only tiles that can be on the border of a stable assembly due to their strength-2 glues. Every tile in the top row is not connected to the rest with the required strength of 4. As depicted, the assembly can only be stable in combination with an additional assembly above it.



(a) (b) (c)

Figure 4.1: (a) Example binary counter that counts up to 4. Each type consists of its own $\mathcal{O}(1)$-sized set of tiles. Single ticks between tiles represent a strength-1 glue, double ticks represent strength-2 glues. (b) Example Turing machine simulation. (c) The form of a computation assembly.



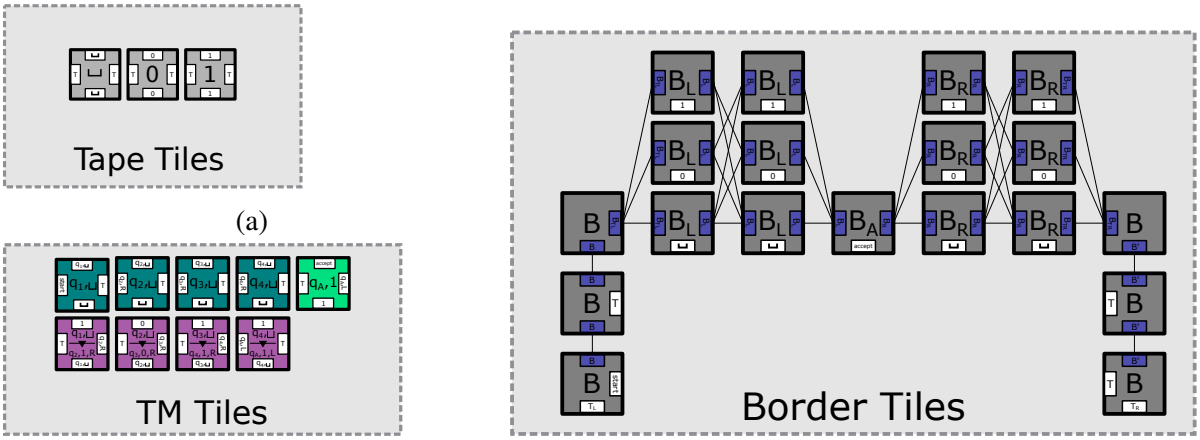
(a) (b)

Figure 4.2: (a) The binary counter consists of $\lceil log_2 k' \rceil$ types of tiles, one for each bit needed to count to k. Each type is a set of 6 tiles. Their generalized glue types of each can be seen in this figure. White colored glue labels represent strength-1, blue colored glue labels represent strength-2. (b) An example of the tile types needed when $k = 4$. There are $\lfloor log_2 k' \rfloor + 1 = 3$ types of tiles. Note the type which represents the most significant bit follows a different pattern than the rest of the types. This is since it must attach to the tiles in the Turing machine simulation section. White colored glue labels represent strength-1, blue colored glue labels represent strength-2. The red number in the glue labels correspond to that tile's type, while the black number corresponds to the values being passed between tiles to achieve the binary counter functionality.

Turing Machine Simulation. We also use known techniques for simulating Turing machines in a self-assembly system [31]. The tiles used for this are shown in Figure 4.3. An example of simulating one step is shown in Figure 4.4a. We use this method to simulate the computation paths of M. Due to the nondeterministic nature of the model, we simulate nondeterministic transition rules by simply having a different tile type for each possible transition (Figure 4.4b). For the instance $(M, k' = 4)$, an example assembly (not stable) that represents an accepting computation path of M is shown in Figure 4.1b. The system created by the reduction also includes the tile set necessary to simulate M in this manner. The set of tiles is disjoint from those used for the binary counter. In the same way, this tile set has the inner tiles (light gray) that perform the computation. These have a strength-1 glue on each side, and border tiles (dark gray). The north border uses a constant number of distinct tile types to ensure that the accept state of the Turing machine must be present in the row below it in order for a stable assembly to be formed.

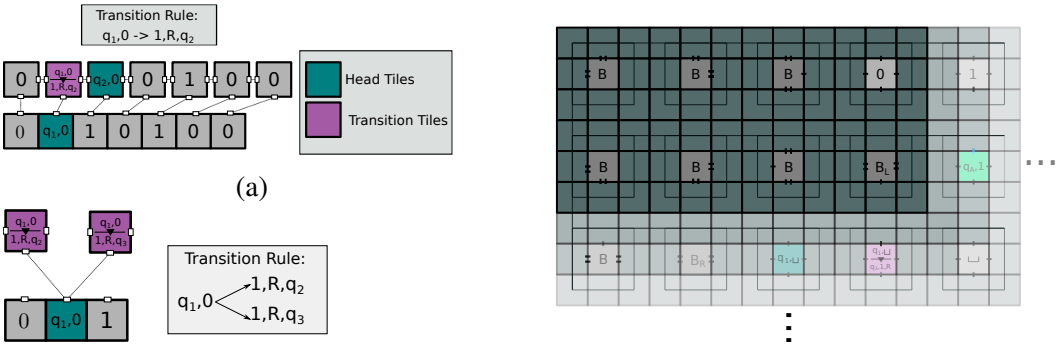
Production of Computation Assembly. The key question of this system is whether the 70 hands can be utilized to bring together ≤ 70 of the described tiles to produce a computation assembly. In the case where the original instance K-A_{NTM} is true, the answer is 'yes'. For the example provided, 28 hands can be used to arrange the tiles of the binary counter section to form an assembly representing the counting from 0 to 4 (Figure 4.1b). Above this, the remaining 42 hands can arrange up to 42 tiles in the Turing machine simulation section in an arrangement that represents an accepting computation path of at most 4 steps (Figure 4.1b). The arrangement of binary counter tiles and Turing machine simulation tiles can form a stable assembly if attached to each other.(Figure 4.1c). Therefore the computation assembly is a producible assembly in a 70-handed system. Note that a computation assembly of size less than 70 can also be produced if there exists an existing computation path strictly less than length 4.

Target Assembly. The target assembly for the K-HAM-UAV instance is an assembly that acts as a "frame" that holds all the tiles previously described (Figure 4.4c). Each tile has a des-



(a) (b) (c)

Figure 4.3: (a) In the Turing machine simulation section, these 3 tiles make up the majority of the tiles used, and represent the current value stored on the tape. The value stored is preserved through the north and south glue types. (b) In the Turing machine simulation section, these tiles represent the head and transitions of the Turing machine. For each state/symbol pair there will be a head tile (dark green), and for each transition rule there will be a transition tile (purple). Note that this set can not represent the full set of tiles created in the reduction, as a properly defined Turing machine would have a transition rule for every state/symbol pair. The tiles shown will be the ones actually used in the example computation assembly. (c) In the Turing machine simulation section, these tiles will form a border around the tiles which represent the computation. White colored glue labels represent strength-1, blue colored glue labels represent strength-2.



(a) (b) (c)

Figure 4.4: (a) One step in the simulation of a Turing machine. The bottom row represents an initial valid Turing machine configuration. The tiles that can attach above this row represent a valid transition to another valid configuration of the Turing machine. (b) Simulation of nondeterministic transition rules. The glues on the south side of both of the transition tiles (purple) are the same, allowing either to be placed above the head tile (green). (c) Example target assembly. Solid lines represent unique strength-4 glues. Every tile type used in the TM simulation section and binary counter section can attach in only one spot.

ignated spot within this assembly that is specified by the frame having the corresponding glues that uniquely identify the tile adjacent to its designated position. Some consideration must be taken in the arrangement of the tiles within this frame to ensure that some extraneous assembly is not built within the frame. Since the tiles that compose the frame have strength-4 glues between them, it is clear that this frame is always built. For every tile in the binary counter section and Turing machine simulation section, there is one spot in the frame that exactly complements its glues, so it is true that each of these tiles can attach to the frame. Thus, the target assembly will always be produced.

4.3.3 Complexity

Given the membership and reduction overview in Sections 4.3.1 and 4.3.2, respectively, we show the following.

Theorem 4.3.3. K-HAM-UAV is complete for coNEXP.

Proof. Lemma 4.3.1 shows that K-HAM-UAV is in the class coNEXP. The reduction shown is a polynomial time reduction from K-A_{NTM} to K-HAM-UAV taking an instance $P = (M, k')$ to an instance $P' = (\Gamma = (T, k, 4), A)$ where $h = (2^{\lceil \log_2(k') \rceil} + 3) \cdot (k' + \lceil \log_2(k') \rceil + 4)$, and $\neg P \iff P'$. It was shown how a true instance of P implies P' is false, through the production of a computation assembly that will never grow into the target assembly.

We now show that the instance P being false implies that the created instance of P' is true. It is clear the target assembly will be produced, but it must be shown that no assembly that is not a subassembly of the target assembly is produced. We first note that the border tiles can not come together alone to form a hollow square. This is because at the points where the border tiles from the binary counter section would meet those from the Turing machine simulation section, there are strength-1 glues (red arrows in Figures 4.1a and 4.1b), meaning the hollow square is not a stable assembly.

From the provided tile types shown, in order to be stable the computation assembly must be enclosed by border tiles (dark gray). Every other tile has only strength-1 glues on every edge, and therefore if the tile were on the border, the assembly would not be stable. Due to a unique glue (shown in green in Figure 4.1a), the right border of the binary counter can only be built if there is a 1 to the left of it in the row representing the most significant bit. Therefore, in order to be stable, the binary counter assembly must have counted up to $2^{\lceil \log_2(k') \rceil}$. Thus, $(2^{\lceil \log_2(k') \rceil} + 3) \cdot (\lceil \log_2(k) \rceil + 2)$ of the allotted hands must be used to "hold" the binary counter in place (28 in our example).

This leaves the system with $(2^{\lceil \log_2(k') \rceil} + 3) \cdot (k' + 2)$ hands left, which can be used to arrange the Turing machine simulation tiles in a way that can attach to the binary counter. Since only the border tiles of the binary counter assembly attach to the border tiles of the simulation assembly, the simulation assembly must be of the same width $(2^{\lceil \log_2(k') \rceil} + 3)$. Thus, it can be at most height $k' + 2$. Since one row has to be used for the top border, the simulation can only utilize $k' - 1$ rows, i.e., k' steps. Due to another unique glue on the top border of the simulation assembly (cyan in Figure 4.1b), the tile B_A can only be stable on an assembly if the row below it contains a tile that represents the accept state. Every tile in the Turing machine simulation section must have a matching glue with all of its neighbors. Since every two adjacent rows in the Turing machine simulation share matching glues, the glue encoding enforces that it is a valid transition from one configuration of the Turing machine to another. Therefore, starting from the initial configuration of M, if there does not exist a computation path that accepts in $\leq k'$ steps, then there is no way to arrange the $k' + 1$ rows in a way that is both stable and has the accept state present. Thus, if the instance P is false, then the only terminal assembly of the created system is the target assembly, so P' is true. □

Corollary 4.3.4. U-K-HAM-UAV is complete for coNP.

Proof. Corollary 4.3.2 shows that U-K-HAM-UAV is in the class coNP. The problem U-K-A_{NTM} where the parameter k denoting the maximum allotted runtime is encoded in unary is coNP-

hard. An equivalent reduction which outputs the same instance $P' = (\Gamma = (T,k,4),A)$ with the difference that k is encoded in unary is a polynomial time reduction from U-K-A_{NTM} to U-K-HAM-UAV. □

CHAPTER V

SUMMARY AND CONCLUSION

In this work we studied the complexity of verification problems in different models of self assembly, each of which being a different generalization of the well studied 2-Handed Assembly Model. We primarily studied two verification problems: Producibility and Unique Assembly Verification Problems.

For the case of 2HAM with prebuilt assemblies, we showed that the problem of verifying whether a system produces an assembly is NP-Complete. Furthermore we proved that the UAV problem is coNP^{NP}-Complete. These results hold even when the prebuilt assemblies are limited to $\mathcal{O}(1)$ size. This shows that the simple capability of starting with small prebuilt assemblies, a capability that is very feasible in real world self assembly implementations, greatly increases the complexity of verifying a systems behavior.

We also studied the Staged Tile Assembly Model. For this work we showed that the problem of Unique Assembly Verification, which in this case is asks if a given assembly is uniquely produced by every bin in the *last* stage of a system. Using a newly introduced technique, covert computation, we proved that the Unique Assembly Verification problem is PSPACE-Complete, and therefore verifying the behavior of Stage Tile Systems, even those promised to be bounded by a certain size, is highly intractable.

Finally we studied the k-Handed Assembly Model, which allows for many assemblies to come together in a single production step. We first showed that even with allowing for this powerful behavior of arbitrary number of assemblies combining in a single step, the producibility

problem admits a polynomial time algorithm. We then showed hardness for two formulations for the Unique Assembly Verification problem. If for a given instance the integer k, denoting the maximum amount of assemblies that can come together in one step, is encoded in binary then the UAV problem is coNEXP-Complete. On the other hand, if the integer k is encoded in unary then the problem is coNP-Complete.

The results here resolve many open problems in the field of self assembly, and provide a more complete picture of hierarchical Self Assembly complexity landscape. The results exemplify how changes to the rules of how these systems behave , even those seemingly small, can greatly affect the computational complexity of verification problems.

BIBLIOGRAPHY

[1] L. ADLEMAN, Q. CHENG, A. GOEL, AND M.-D. HUANG, *Running time and program size for self-assembled squares*, in Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing, STOC '01, New York, NY, USA, 2001, Association for Computing Machinery, p. 740–748.

[2] G. AGGARWAL, Q. CHENG, M. H. GOLDWASSER, M.-Y. KAO, P. M. DE ESPANES, AND R. T. SCHWELLER, *Complexities for generalized models of self-assembly*, SIAM Journal on Computing, 34 (2005), pp. 1493–1515.

[3] N. ALON, R. YUSTER, AND U. ZWICK, *Finding and counting given length cycles*, Algorithmica, 17 (1997), pp. 209–223.

[4] D. CABALLERO, T. GOMEZ, R. SCHWELLER, AND T. WYLIE, *Verification and computation in restricted tile automata*, in 26th International Confrence on DNA computing and Molecular Programming, 2020.

[5] D. CABALLERO, T. GOMEZ, R. SCHWELLER, AND T. WYLIE, *Unique assembly verification in two-handed self-assembly*, in International Colloquium on Automata, Languages, and Programming, Springer, 2022, p. To Appear.

[6] S. CANNON, E. D. DEMAINE, M. L. DEMAINE, S. EISENSTAT, M. J. PATITZ, R. T. SCHWELLER, S. M. SUMMERS, AND A. WINSLOW, *Two Hands Are Better Than One (up to constant factors): Self-Assembly In The 2HAM vs. aTAM*, in 30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013), vol. 20 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, pp. 172–184.

[7] A. A. CANTU, A. LUCHSINGER, R. SCHWELLER, AND T. WYLIE, *Covert computation in self-assembled circuits*, Algorithmica, 83 (2021), pp. 531–552.

[8] A. A. CANTU, A. LUCHSINGER, R. SCHWELLER, AND T. WYLIE, *Covert Computation in Self-Assembled Circuits*, Algorithmica, 83 (2021), pp. 531–552. arXiv:1908.06068.

[9] C. T. CHALK, D. A. FERNANDEZ, A. HUERTA, M. A. MALDONADO, R. T. SCHWELLER, AND L. SWEET, *Strict self-assembly of fractals using multiple hands*, Algorithmica, 76 (2016), pp. 195–224.

[10] H.-L. CHEN AND D. DOTY, *Parallelism and time in hierarchical self-assembly*, Computing Research Repository - CORR, (2011).

[11] E. D. DEMAINE, M. L. DEMAINE, S. P. FEKETE, M. ISHAQUE, E. RAFALIN, R. T. SCHWELLER, AND D. L. SOUVAINE, *Staged self-assembly: nanomanufacture of arbitrary shapes with o (1) glues*, Natural Computing, 7 (2008), pp. 347–370.

[12] D. DOTY, *Theory of algorithmic self-assembly*, Communications of the ACM, 55 (2012), pp. 78–88.

[13] ——, *Producibility in hierarchical self-assembly*, (2014), pp. 142–154.

[14] M. ENDO, T. SUGITA, Y. KATSUDA, K. HIDAKA, AND H. SUGIYAMA, *Programmed-assembly system using DNA jigsaw pieces*, Chemistry: A European Journal, (2010), pp. 5362–5368.

[15] D. EPPSTEIN, *Subgraph isomorphism in planar graphs and related problems*, Journal of Graph Algorithms and Applications, 3 (1999), pp. 1–27.

[16] C. EVANS, *Crystals that Count! Physical Principles and Experimental Investigations of DNA Tile Self-Assembly*, PhD thesis, California Inst. of Tech., 2014.

[17] S. P. FEKETE, J. HENDRICKS, M. J. PATITZ, T. A. ROGERS, AND R. T. SCHWELLER, *Universal computation with arbitrary polyomino tiles in non-cooperative self-assembly*, in Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2014, pp. 148–167.

[18] B. FU, M. J. PATITZ, R. T. SCHWELLER, AND R. SHELINE, *Self-assembly with geometric tiles*, in International Colloquium on Automata, Languages, and Programming, Springer, 2012, pp. 714–725.

[19] T. J. FU AND N. C. SEEMAN, *Dna double-crossover molecules*, Biochemistry, 32 (1993), pp. 3211–3220.

[20] Y. HE, T. YE, M. SU, C. ZHANG, A. E. RIBBE, W. JIANG, AND C. MAO, *Hierarchical self-assembly of dna into symmetric supramolecular polyhedra*, Nature, 452 (2008), pp. 198–201.

[21] A. KEENAN, R. SCHWELLER, M. SHERMAN, AND X. ZHONG, *Fast arithmetic in algorithmic self-assembly*, Natural Computing, 15 (2016), pp. 115–128.

[22] Ł. KOWALIK, *Short cycles in planar graphs*, in Graph-Theoretic Concepts in Computer Science, H. L. Bodlaender, ed., Berlin, Heidelberg, 2003, Springer Berlin Heidelberg, pp. 284–296.

[23] D. MINEV, C. M. WINTERSINGER, A. ERSHOVA, AND W. M. SHIH, *Robust nucleation control via crisscross polymerization of highly coordinated dna slats*, Nature communications, 12 (2021), pp. 1–9.

[24] R. O'GRADY, R. GROSS, A. L. CHRISTENSEN, AND M. DORIGO, *Self-assembly strategies in a group of autonomous mobile robots*, Autonomous Robots, 28 (2010), pp. 439–455.

[25] M. J. PATITZ, *An introduction to tile-based self-assembly and a survey of recent results*, Natural Computing, 13 (2014), pp. 195–224.

[26] P. W. ROTHEMUND AND E. WINFREE, *The program-size complexity of self-assembled squares*, in Proceedings of the thirty-second annual ACM symposium on Theory of computing, 2000, pp. 459–468.

[27] M. RUBENSTEIN, A. CORNEJO, AND R. NAGPAL, *Programmable self-assembly in a thousand-robot swarm*, Science, 345 (2014), pp. 795–799.

[28] R. SCHWELLER, A. WINSLOW, AND T. WYLIE, *Complexities for high-temperature two-handed tile self-assembly*, in DNA Computing and Molecular Programming, R. Brijder and L. Qian, eds., Cham, 2017, Springer International Publishing, pp. 98–109.

[29] ——, *Verification in staged tile self-assembly*, Natural Computing, 18 (2019), pp. 107–117.

[30] N. C. SEEMAN, *Nucleic acid junctions and lattices*, Journal of Theoretical Biology, 99 (1982), pp. 237–247.

[31] E. WINFREE, *Algorithmic Self-Assembly of DNA*, PhD thesis, California Institute of Technology, June 1998.

[32] S. WOO AND P. W. ROTHEMUND, *Stacking bonds: Programming molecular recognition based on the geometry of DNA nanostructures*, Nature Chemistry, 3 (2011), pp. 620–627.

[33] D. WOODS, *Intrinsic universality and the computational power of self-assembly*, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 373 (2015), p. 20140214.

[34] D. WOODS, D. DOTY, C. MYHRVOLD, J. HUI, F. ZHOU, P. YIN, AND E. WINFREE, *Diverse and robust molecular algorithms using reprogrammable dna self-assembly*, Nature, 567 (2019), pp. 366–372.

[35] D. WOODS, D. DOTY, C. MYHRVOLD, J. HUI, F. ZHOU, P. YIN, AND E. WINFREE, *Diverse and robust molecular algorithms using reprogrammable dna self-assembly*, Nature, 567 (2019), pp. 366–372.

[36] C. ZHANG, R. J. MACFARLANE, K. L. YOUNG, C. H. J. CHOI, L. HAO, E. AUYEUNG, G. LIU, X. ZHOU, AND C. A. MIRKIN, *A general approach to dna-programmable atom equivalents*, Nature Materials, 12 (2013), pp. 741–746.

BIOGRAPHICAL SKETCH

David Eric Caballero Jr. was born and primarily raised in the Rio Grande Valley, and attended high school at the Science Academy of South Texas. Here where he developed an interest in computers, primarily in programming through the introduction to Computer Science classes.

He attended the University of Texas Rio Grande Valley pursing a Bachelor's Degree in Computer Science. During his undergraduate studies he was introduced to theoretical computer science research through the Algorithmic Self Assembly Research Group and quickly grew an interest. He performed research in the fields of algorithmic self assembly and particle motion planning under Dr. Robert Schweller and Dr. Tim Wylie. Through this he was able become achieve his first publication during his undergraduate career. He earned his Bachelor's of Science in Computer Science in May of 2020, graduating *magna cum laude*, and then went on to pursue a Master of Science degree in Computer Science at the University of Texas Rio Grade Valley. Through collaboration with the research group he was able to solve open problems, attend and present at research conferences, and coauthored 14 peer reviewed publications by his graduation in May of 2022. David can be contacted at davidcaballeroe@gmail.com.